Switching Considerations in Storage Networks

by

Leung Yiu Tong

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Philosophy

in

Information Engineering

The Chinese University of Hong Kong

August 2003

Content

Abstract

Sharing and distribution of business/financial data through computer networks have become a critical issue as corporations grow. Over the years, many proposals of industrial standards have been published on *storage networks*, covering both the network topology and the network protocol but yet there is no final version that is good for all users. In this thesis, we investigate the pros and cons of various proposals, compare their similarities and differences, make amendments for optimization, and forecast the future use of storage networks.

Upon the introduction of fiber optics to data communication, switching has become a crucial limitation to system performance. Investigation has been made on switching designs as well as switching functionalities in storage networks. Original ideas are proposed in order to provide better service and support newer technologies. Amendments and additional services are built on top of *crossbar switching*, *shared buffer memory switching* and *algebraic switching*. The new designs are then integrated with existing technologies to support special network features of storage networks.

內容撮要

企業發展蓬勃，商務及財經資訊的傳遞成爲業務發展中重要的一環。近年，市場上出現了不少關於存儲網絡（*storage network*）架構和通訊協定的建議書，但至今尚未有一項建議能爲每一位使用者提供完善周全的服務。本論文將詳述不同建議的優劣，比較相互的同異，提出改善表現的修訂，並展望業界的發展方向。

自從引用光纖作訊息傳遞，訊息交換便成爲整體表現的瓶頸。本論文研究存儲網絡的交換設計及網絡服務，並提出了有關的原建議，以提供更佳的服務及支援更新技術。此等改良及附加服務建基於交叉開關交換（*crossbar switching*）、共享緩存交換（*shared buffer memory switching*）及代數交換（*algebraic switching*），綜合現存技術以支援獨特的存儲網絡應用程式。

Acknowledge

I would like to thank my advisor, Professor S.-Y. Robert Li, who has taught me lots of useful material and instructed me to learn and work independently during my two years of Master of Philosophy studies.

I would also like to thank Department of Information Engineering of The Chinese University of Hong Kong for admitting me to the research programme. Under the research atmosphere of the department, I have broadened my knowledge in various aspects in communications. Through the two years of research studies, I have learnt not just the academic knowledge but also the way of learning, which will be beneficial to my living in this modern world.

1. Introduction

1.1 Motivation

Networking in business sector has been developing for many years. At first, the main functionality was for the flow of business information or financial data. As business grew, sharing of information within an organization becomes part of the daily life. In the old days, file sharing was by passing around floppy disks or decks of documents.

Through the advancement of networking, file sharing becomes more natural and convenient. In current technologies, the easiest and cheapest way of sharing data via a network is to adopt the general networking functionalities provided by desktop computers. However, this approach may not be the best way for corporate users. Any heavy network traffic, such as backup processes, can block the whole corporation from network access. *Storage networks* are deployed to provide solutions to such problem. However, despite the variety of proposals to both the network topologies and protocols, there is yet no single proposal that is perfect to all users **[1]**.

Due to the scalability in terms of storage size, more corporations have turned into storage networks. Storage-intensive applications, such as *video-on-demand (VoD)* and online cyber mall are also foreseeable. In storage networking, transmission efficiency and storage capacity are the two main concerns of the users. Different designs on topologies and protocols make it easy to expand

storage capacities, but optimization on networks efficiency is rare. As storage networks become more distributed and the improvement on transmission speed, a simple time-shared bus is no longer adequate for switching and I/O purpose. The switching issue in storage networking is getting important and worth investigating.

This thesis is intended for a bridge between storage networks and switching. Detailed discussion has been made on the area of storage network topology and protocol, including various proposals to industrial standards. Special considerations on the switching deployment in storage networks are carried out. In particular, investigations are put into the issues of

1. Choosing a Suitable Switch
2. *Quality of Service (QoS)*
3. Multicasting
4. Load Balancing
5. Optimization on Storage Utilization

Switching technologies have also been reviewed in this thesis, giving qualitative investigation and comparison among different technologies. Specific improvements have been made on the switching technologies for optimization for use in storage networks and its applications. I hope that this thesis can give readers an insight of the importance of switching issue in storage networking.

1.2 Thesis Organization

The thesis is divided into five chapters, which covers fundamental concepts of storage networking, switching technologies, investigation in switching issues in storage networks and original proposals on amendments in corresponding switching designs.

As the overall introduction, Chapter 1 presents the broad idea of the subject. Chapter 2 reviews fundamental knowledge of storage networks, including industrial standard proposals on topologies and protocols. Chapter 3 goes through the basic switching technologies and examines the possibility of use in storage networks. New ideas have been proposed in this chapter to fine-tone the technologies into the use of storage networks. Chapter 4 serves as the core linkage between switching and storage networking via qualitative comparisons and makes suggestions on the switching issue in storage networks. Chapter 5 summarizes the main concepts in this thesis, presenting opinion in future development trend in storage networking. Besides the conclusion, it also provides a review on original contributions of this thesis.

2. Storage Network Fundamentals

Storage networks are arisen from the problem of information sharing. In order to facilitate internal spreading of data, an information database with network capability is essential.

Currently, most network applications are deployed from the server-client model, where the servers can be the bottleneck of performance, or the single point of failure. Multi-server approach can be an alternative, but it can easily add load to the network traffic. Moreover, unbalanced utilization can be a challenge to network administrators.

"Storage Networking is the movement of data across time and space simultaneously **[2]**." A root problem of networking is the network topology and its underlying protocols. In this chapter, we will be investigating into various proposals in storage networking, including network topology and protocols. Collective evaluations on these proposals are also presented as the conclusion of this chapter.

2.1 Storage Network Topology

Network topology refers to the way of interconnecting network devices. In storage networks, the devices involved includes storage devices, servers, clients, switches, routers, and sometimes gateways. Different ways in connecting the

devices give different properties to the network.

There are three common types of storage network topology, namely *Direct Attached Storage* (DAS), *Network Attached Storage* (NAS) and *Storage Area Network* (SAN). In this section, we will briefly introduce the fundamental concepts of these three topologies and explain the differences between them.

2.1.1 Direct Attached Storage (DAS)

Direct Attached Storage is the fundamental concept of Storage Network. What it does is to install storage devices on network hosts, where these hosts can be freely assessed throughout the network. Figure 2.1 shows a sample setup of DAS. Storage devices in DAS can be directly connecting to the server by external interface, or embedded inside the server **[3, 4]**. One of the most popular interfaces is the *Small Computer System Interface* (SCSI).

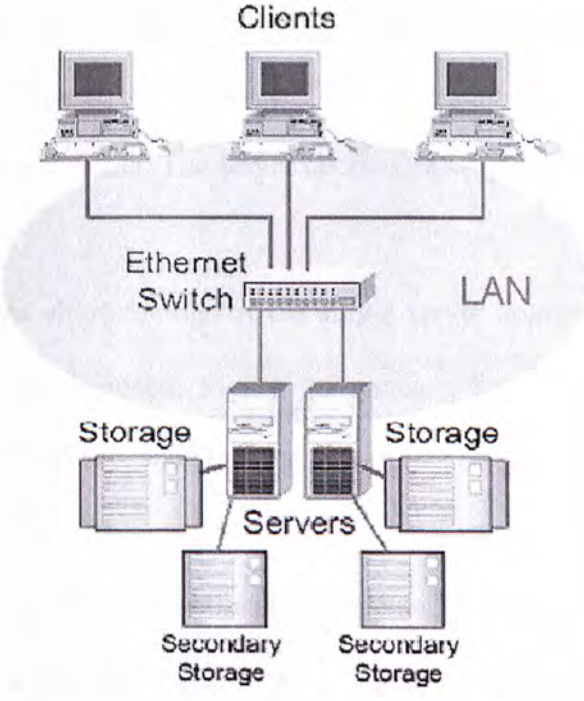


Figure 2.1 Direct Attached Storage (DAS)

Direct Attached Storage is the simplest way of setting up a storage network. Network users can log into the storage server and get the required data. This technology is widely deployed in the Microsoft Windows Network Clients.

The communication between the server and the storage devices is basically depending on the local file system. Block I/O command is usually used in these communications. Raw data chunks (instead of files) are transmitted from the storage devices to the server, and then to the clients. It can be the application itself initiates a block-level request to the storage devices, or the application initiates a file request, and the file system in turn initiates a block request to the devices.

Although DAS is the easiest approach in constructing a storage network, it has quite a number of disadvantages. Since all of the storage devices are connected to server through SCSI interface, and the number of SCSI interfaces available to a server is limited, storage capability is limited. Therefore, this setting is not scalable. Moreover, if the server fails, it is impossible for any client to assess the corresponding data. The server becomes a single point of failure.

To leverage the shortcomings of the single server approach, a multi-servers variation of DAS is proposed. Figure 2.2 gives a brief concept to distribute problems to multiple servers such that the failure rate drops.

Apart from the above-mentioned problems, network backup is another challenging issue. In DAS, backup takes up computation time of server CPU as well as network bandwidth. This makes data sharing difficult. The limited

working distance is also one of the shortcomings faced by DAS.

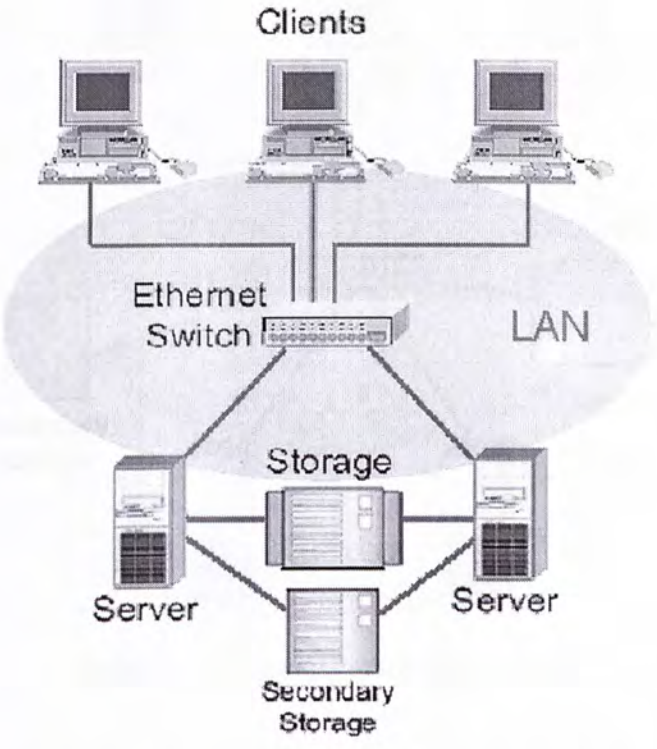


Figure 2.2 Direct Attached Storage with multi-servers

2.1.2 Network Attached Storage (NAS)

Network Attached Storage is another proposed network topology for storage networks. Instead of putting the storage devices at general-purpose servers, NAS suggests to connect these devices to the network directly as a network host, known as a NAS device. NAS device is a dedicated, high performance, standalone storage device that function at high speeds. It is platform-independent and it can support a wide range of clients **[2, 4]**. Figure 2.3 gives an overview of NAS structure.

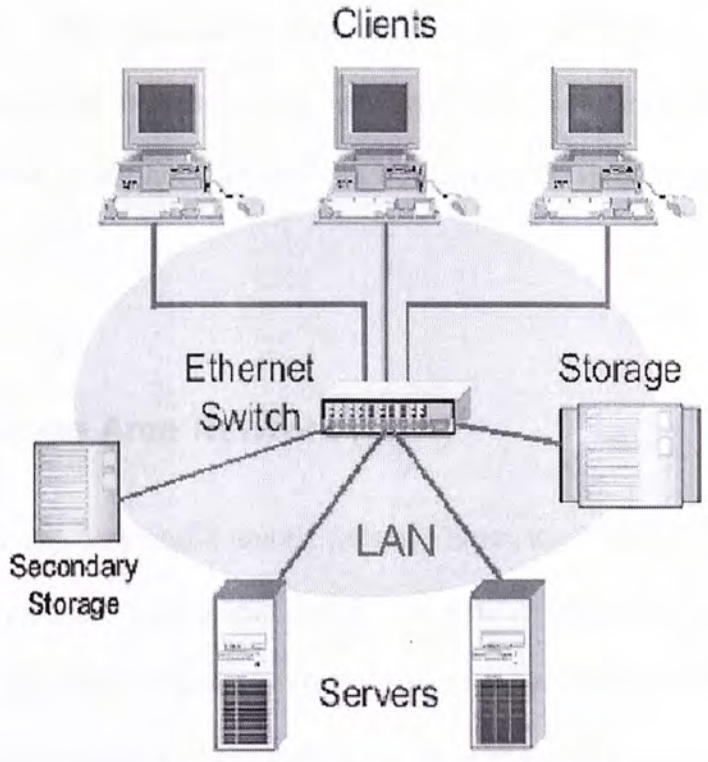


Figure 2.3 Network Attached Storage (NAS)

In contrast to DAS, NAS adopts file I/O approach. Whenever clients request for information, NAS server sends a file request to NAS device. The file system of the NAS device processes the file request, translate it into block level request and locate the suitable data. The data is then transferred to clients in form of a file, and therefore the NAS devices are also known as a *filer*.

Due to the separation of data from the servers and the localization of specialized NAS devices, the overall network productivity is increased and the storage-related problems are lessened. Also, this technology can improve the server performance as the strain of file processing is eliminated from the server.

However, implementing NAS solutions on a network also has several shortcomings. The interaction in processing data transfer is slow and it generates

more network traffic. This makes NAS only suitable for small to medium scale networks, but not for extremely large networks. Also, since the NAS devices are directly attached to the network and can be easily accessed, the vulnerability is increased.

2.1.3 Storage Area Network (SAN)

As the heavy data traffic among network hosts, such as that among storage devices, servers and clients can block the whole production network from functioning, experts have looked into the possibility to build a network which is isolated form the general purpose network. This storage-only network is known as Storage Area Network (SAN).

According to Storage Network Industry Association (SNIA), SAN is "a network whose primary purpose is the transfer of data between computer systems and storage elements and among storage elements **[5]**." SAN is a technology developed to handle enormous amounts of data securely and reliably without hampering overall network performance. The storage devices are directly connected to neither the servers nor the network clients. All the storage devices are interconnected to each other to form a separate network, which can be accessed only through the servers **[3, 4, 6, and 7]**. A typical SAN structure is given in Figure 2.4.

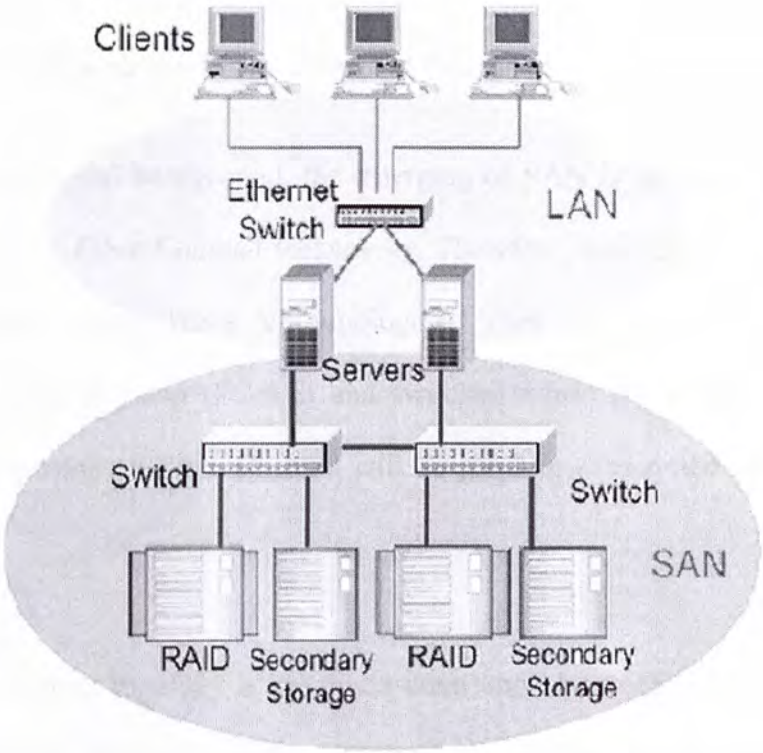


Figure 2.4 Storage Area Network (SAN)

The file system of SAN resides in the SAN servers, and therefore it primarily uses Block I/O. Just like that of DAS, they are initiated mainly by direct block request in application level, or block request triggered by file request in the file system.

SAN has scaleable performance and its infrastructure expands easily for on-demand storage. It supports multi-paths for redundancy, fault tolerance and performance so that it can increase data integrity. The extended distance and consolidated storage resources are also the strengths of SAN. In this case, data can be accessed from anywhere at anytime and the total cost of storage ownership is reduced.

2.1.3.1 SAN and the Fibre Channel Protocol

From historical background, the emerging of SAN is strongly based on the development of *Fibre Channel* technology. Therefore, similar to Fibre Channel, SAN supports three types of topologies. They are point-to-point, Fibre Channel-Arbitrated Loop (FC-AL) and switched fabric **[2, 3, and 7]**. A more detailed discussion on Fibre Channel will be given in subsequent section of this chapter.

Point-to-point topology is the direct connection between two SAN devices. Because of the dedicated nature of physical connections, the point-to-point topology is the fastest, simplest and easiest to implement and manage. However, the point-to-point topology is not commonly used to build an entire storage network because it is the costliest of the three.

FC-AL topology can support up to 127 nodes and devices simultaneously. However, since the link bandwidth is shared among connected devices in this topology, the performance can degrade considerably if all of the other 126 nodes are connected to the same node. Also, due to the shared nature of the loop, node loops need to arbitrate for loop control. After a node gains control of the loop, only one loop node can transmit data at a time. So this topology should be used in a SAN if the number of nodes is not high or transmissions are not time-sensitive.

Switched fabric topology consists of an interconnection of switches that can support a staggering 16 million nodes, as shown in Figure 2.5. To connect local and remote storage interfaces to different subnets, SAN interconnection devices,

which are known as SAN interconnects, are introduced. Commonly used SAN interconnects include host bus adapters (HBA), hubs, bridge, router, fabric switches and gateways.

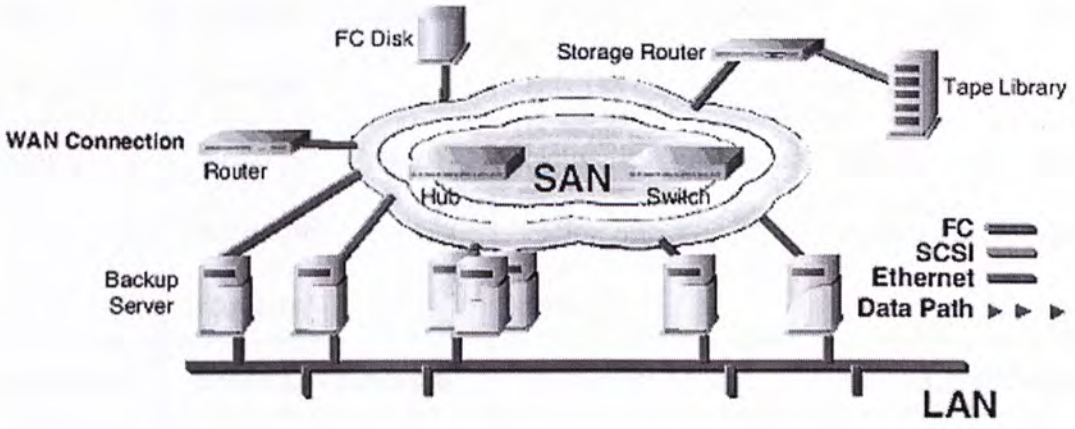


Figure 2.5 Switched fabric SAN

The high point of the switched fabric topology is that despite the addition of devices to the fabric, the aggregate bandwidth of the topology increases because switches that form the backbone of the network are indeed high-performing, non-blocking devices.

2.1.4 Summary on Storage Network Topology

Among the three topologies introduced in this section, they do share a few similarities, but actually they have quite a few differences. Comparisons on these two technologies are tabulated below **[3, 7]**:

	DAS	NAS	SAN
Network Topology	Access to files by general purpose servers	Access to files on a specialized file server	Network connectivity between systems and storage
Network Nature	General purpose network	General purpose network	High-speed, storage-only network
Protocol	Network Protocol transmitting file requests	Network protocols redirecting file requests	Specialized storage network protocols
I/O Command	Block Level Access	File Level Access	Block Level Access
Clients Supported	Heterogeneous	Homogeneous	Heterogeneous
Storage Management	Server-side	Server-side	Network-side

Table 2.1 Comparison on DAS, NAS and SAN

The co-relationship between network and file system of the three mentioned network topologies are also of research interest. In DAS, the server runs application to accept file request from network users, initiate file request to the file system, and then initiate block request to the storage. NAS server application receives file request from clients, and then redirect them through the network to NAS devices. SAN server application processes file requests, convert them into block request by their file system, and transmit them to the storage devices. A conceptual picture is given in Figure 2.6.

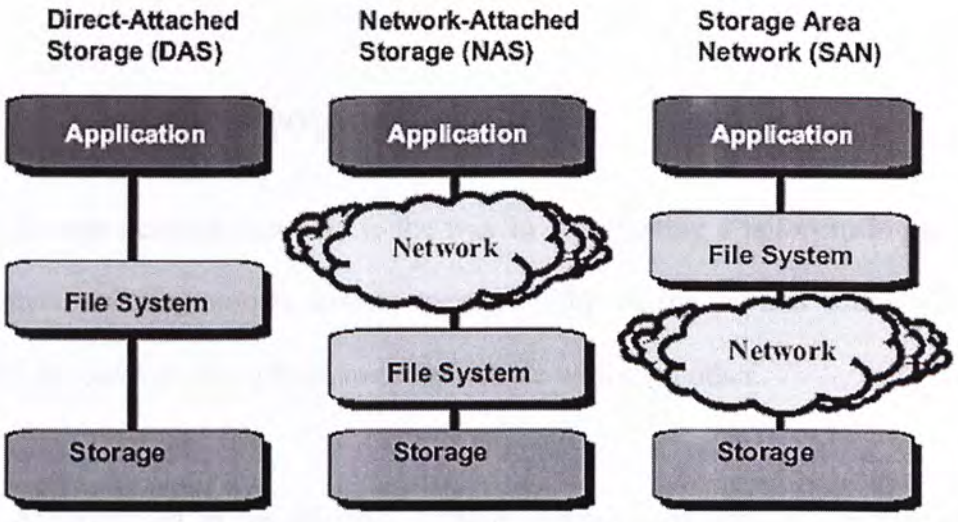


Figure 2.6 Relationship of Network and File System in Storage Networks

Despite the difference of the network models for storage networking, neither of them precludes the existence of the others. The technologies can actually co-exist to provide a more suitable solution to enterprise. Figure 2.7 gives an example of a hybrid type storage network. SAN acts as an infrastructure for NAS, where NAS exist as "front end" while SAN exists as "back end".

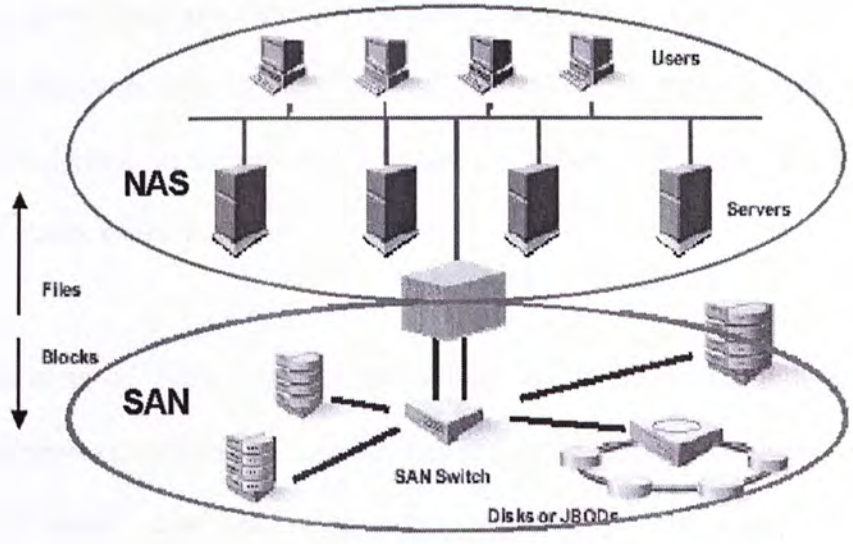


Figure 2.7 Co-existence of SAN and NAS

2.2 Storage Protocol

Storage network topology is the way in constructing a tailor-made network by interconnecting storage devices together. With all the devices connected, we need a protocol enabling them to communicate with each other.

As discussed in the previous section, some of the network models adopt generic network protocols, while some of them are targeted at specialized protocols. In this section, we will be discussing some of the popular storage network protocols, which are suitable for use in different topologies.

2.2.1 Fibre Channel

Fibre Channel is a fast and highly-reliable interconnect technology that allows concurrent communications among workstations, mainframes, servers, data storage systems and other peripherals to meet the needs of a data center. It provides interconnection systems for multiple topologies that can scale to a total system bandwidth on the order of a terabit per second **[8]**. Figure 2.8 shows the frame structure of the protocol.

The aims of Fibre Channel technology are to facilitate high-speed data transfers between servers, storage devices and other network devices; provides a high-performance and yet inexpensive solution, which does not lead to skyrocketing of implementation costs. It also provides a highly mature infrastructure that responds well to future growths and advancement by providing

a generic solution that supports the heterogeneous environments seamlessly and reuses existing protocols and infrastructures.

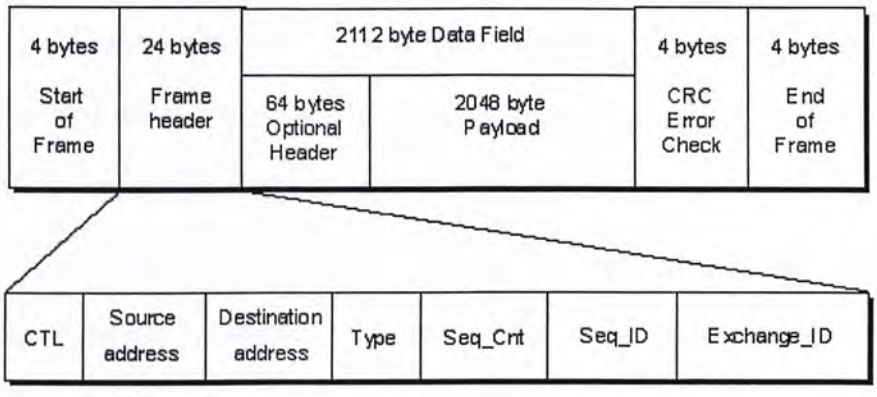


Figure 2.8 Frame structure of Fibre Channel Protocol

Fibre Channel supports different types of traffic, which covers the physical, link-state, network and transport layer of the OSI model **[9]**. Class 1 is the dedicated connection service. This is an acknowledged connection-oriented delivery. When a Class 1 connection has been established between two devices, no other devices can share the engaged link. Class 2 is a connectionless, acknowledged multiplexed service. As with traditional packet-switched systems, the path between two ports is not dedicated, allowing for shared use of the link's bandwidth. Class 3 is a connectionless, unacknowledged datagram service. Due to this traffic nature, the device driver has to determine if data is not receives and requests for re-transmission. Class 4 is a virtual circuit service, where fractional bandwidth is reserved and transmission sequence order is preserved for the quality of service.

Fibre Channel has been developed for several years, and the technology is getting mature. There are different types of devices equipping Fibre Channel as a

supporting protocol, and enterprises have been deploying it to their back-end storage protocol. However, the working distance of Fibre Channel is limited as it relies on its own fiber optical network. It is nearly impossible to link the network of the Asia headquarter with the America headquarter! Due to this fact, experts have proposed the use of WAN as an interconnecting tool, and raised different proposals in supplement to the original version.

Fibre Channel is regarded as one of the most viable solutions for IT professionals who need reliable, cost-effective information storage and delivery at blazing speeds **[7, 8]**. Fibre Channel mainly adopts fiber optics as their transmission medium, solely because of the need of speedy data synchronization. After the introduction of Gigabit Ethernet, the strength of Fibre Channel is declining. Incorporates turn to dig into the possibility of extend existing network infrastructure to storage purpose. Even thought the upcoming of IP storage protocols, enterprises will not stop the use of Fibre Channel, as they have already invested a lot in setting up the system, which is still working fine. Fibre Channel will not disappear any time soon. Instead, it will continue to grow and expand for this new millennium.

2.2.1.1 Fibre Channel over IP (FCIP)

FCIP is a TCP/IP-based tunneling protocol for connecting geographically distributed Fibre Channel SANs transparently to both FC and IP. It relies on IP-based network services to provide the connectivity between the SAN islands over LANs, MANs, or WANs. Figure 2.9 shows a typical usage in FCIP. FCIP relies upon TCP for congestion control and management and upon both TCP and

FC for data error correction and data loss recovery **[3, 4, and 7]**.

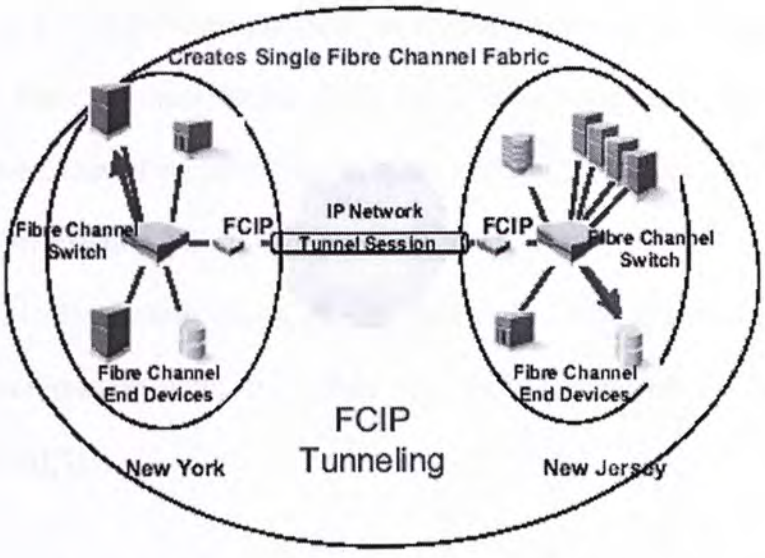


Figure 2.9 Connecting Fibre Channel "Islands" with FCIP

The implementation of FCIP technology requires the use of FCIP devices. These devices encapsulate Fibre Channel frames into TCP segments during transmission. At the receiver end, FCIP devices re-assemble the received TCP segments into original Fibre Channel frames. FCIP devices are available as standalone devices or they can be integrated with IP devices, such as IP routers and switches.

FCIP protocol is a powerful networking technology and offers high-speed and highly reliable data transfers over long distances. It is cost effective as it can be implemented with minimum changes to the existing infrastructure. Besides, FCIP allows the use of existing SAN management applications, which highly reduces the management cost of storage islands across an IP-based infrastructure.

2.2.1.2 Internet Fibre Channel Protocol (iFCP)

iFCP is a TCP/IP-based protocol for interconnecting Fibre Channel storage devices or Fibre Channel SANs using an IP infrastructure to complement or replace Fibre Channel switching and routing elements. This protocol enables the attachment of existing FC storage products to an IP network. iFCP extends the existing FC infrastructure across IP networks and allows IP based core storage fabrics in conjunction with, or in place of, Fibre Channel fabrics, as shown in Figure 2.10 **[4][7]**.

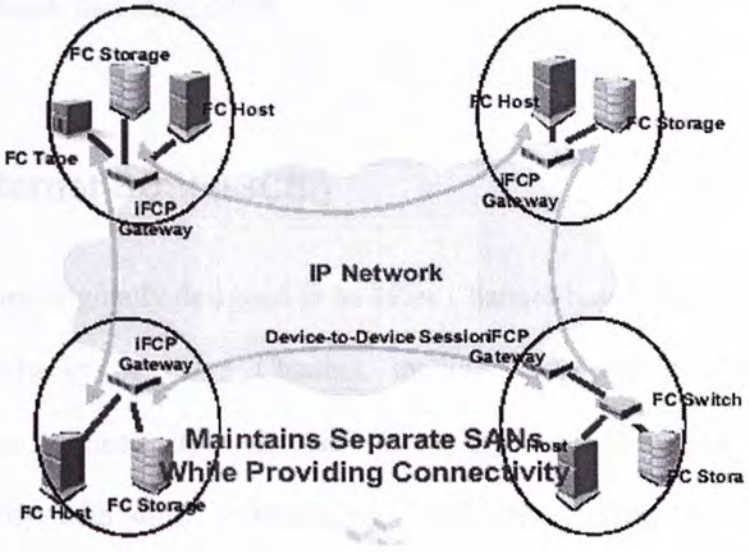


Figure 2.10 Interconnecting SANs with iFCP

In iFCP, each SAN island can control its own address space. Locally attached devices trying to reach remote devices are presented with a translated address that matches the local network range. This allows multiple SAN islands to reach the same remote device using different virtual addresses if any address space conflict exists **[10]**.

iFCP acts as a technology catalyst behind networking products that complement and enhance the functionality of FC devices and networks. It allows users to interconnect FC devices across TCP/IP networks of any distance and enables interconnections to existing FC SANs, including FC switched, directors, host bus adapters and storage subsystems. It offers highly-scalable implementations that provide the flexibility of managed name servers across interconnected FC SAN fabrics and provides robust mechanisms for Fibre Channel SANs to be interconnected as autonomous regions. Also, it maximizes the use of IP networking and provides a migration path from pure FC SANs to Internet Protocol, including iSCSI.

2.2.2 Internet SCSI (iSCSI)

SANs are originally designed to be Fibre Channel-based. However, there are certain drawbacks of Fibre Channel, include high cost and difficulty in implementation, necessity of staff retraining or additional staff, and incompatibility with other technologies. These shortcomings have led to the emergence of iSCSI.

iSCSI is a TCP/IP-based protocol for establishing and managing connections between IP-based storage devices, hosts and clients. It provides end-to-end native IP storage and provides a new mechanism for encapsulating SCSI commands on an IP network **[3, 4]**. Network end nodes, such as servers and storage devices, act as the basic iSCSI devices. In the iSCSI terminology, servers that request data transfers and encapsulate SCSI commands into TCP/IP packets are referred to as

iSCSI initiators. Storage devices that receive iSCSI commands and exchange data over an IP-based LAN or WAN are known as iSCSI targets. Figure 2.11 suggests a way to adopt iSCSI as the building block of IP SANs.

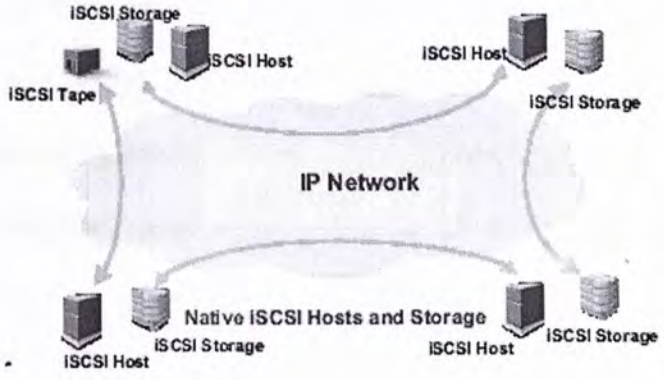


Figure 2.11 iSCSI SANs

In iSCSI, the demarcation for transporting information is at a very high level compared to FCIP or iFCP. SCSI commands such as read and write are reused and concepts such as Logical Units (LUs) are maintained, but all frame format and header information is newly constructed from the top down with a focus on IP and IP addressing **[10]**.

iSCSI data transfers are connection-oriented. To exchange data and commands with the target, a session must be established between the initiator and target LUs. A session is a set of one or more TCP connections, which carry control messages, SCSI commands, parameters, and iSCSI Protocol Data Units (PDUs) between the two communicating LUs.

iSCSI is a powerful networking technology. However, there are still several

challenges iSCSI has to overcome. As most of the SANs today have been deployed in Fibre Channel already, extra investment is needed to migrate from pure Fibre Channel environment to iSCSI or hybrid environment. Also, the SCSI protocol demands data integrity while data transmission in IP networks is unreliable. As the switching in IP network is in per packet basis, end-to-end delay can highly vary, and quality of service (QoS) control is difficult. Moreover, the overhead in processing packet headers is high when we adopt TCP over iSCSI for flow control, therefore it may not be so efficient for some of the applications.

2.2.3 InfiniBand

InfiniBand is a new I/O interconnection technology. It scales as a network switch and provides a mechanism to share I/O interconnects among many servers. InfiniBand is designed for server clusters and I/O for storage networks. It abandons the shared-bus concepts of the *Peripheral Component Interconnect* (PCI) bus, shifting I/O control from processors to a channel-based, switched-fabric, point-to-point, full-duplex interconnect architecture.

Traditionally, PCI bus is the dominant bus used in both desktop and server machines for attaching I/O peripherals to the CPU/memory complex. Even today's powerful desktop machines have lots of capacity available with the PCI bus in the typical configuration, server machines are starting to hit the upper limits of the shared bus architecture.

InfiniBand breaks through the bandwidth limitations of the PCI bus by migrating from the traditional shared bus architecture into switched fabric

architecture **[11]**. The simplest configuration of an InfiniBand installation is to connect two or more nodes to one another through the fabric. A node can be either a host device such as a server or an I/O device such as RAID subsystem. The fabric itself may consist of a single switch in the simplest case, or a collection of interconnected switches and routers. Each node connects to the fabric through two types of channel adapters, Host Channel Adapters (HCA) or Target Channel Adapters (TCA). Figure 2.12 shows a sample setup of the InfiniBand Architecture.

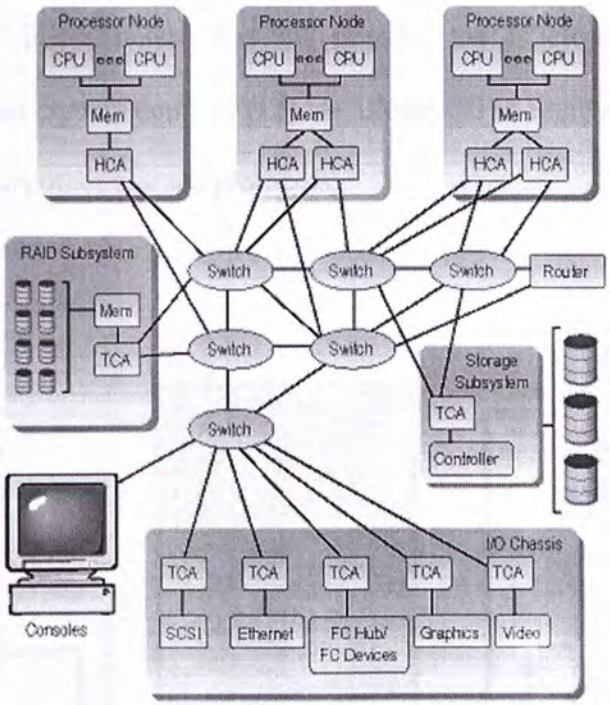


Figure 2.12 InfiniBand Architecture

InfiniBand defines four types of devices: HCA, TCA, switches and routers. The HCA is installed in the server and connects to one or more switches. I/O devices connect to the switches through a TCA, thereby creating a subnet with up to 64,000 nodes. The HCA can communicate with one or more TCAs either

directly or through one or more switches. A router interconnects several subnets. These components transform the system bus into a universal, dynamically configured, scalable interconnection mechanism that enables computers, peripherals and networks to work cohesively to form one enormous, hybrid system.

InfiniBand Architecture (IBA) does not eliminate the need for other existing interconnection technologies. Indeed, it creates a more efficient way to connect storage and communication networks and server clusters together, while delivering an I/O infrastructure that will produce the efficiency, reliability and scalability that data centers demand **[12]**. As illustrated in Figure 2.13, IBA can be extended to support other storage protocols.

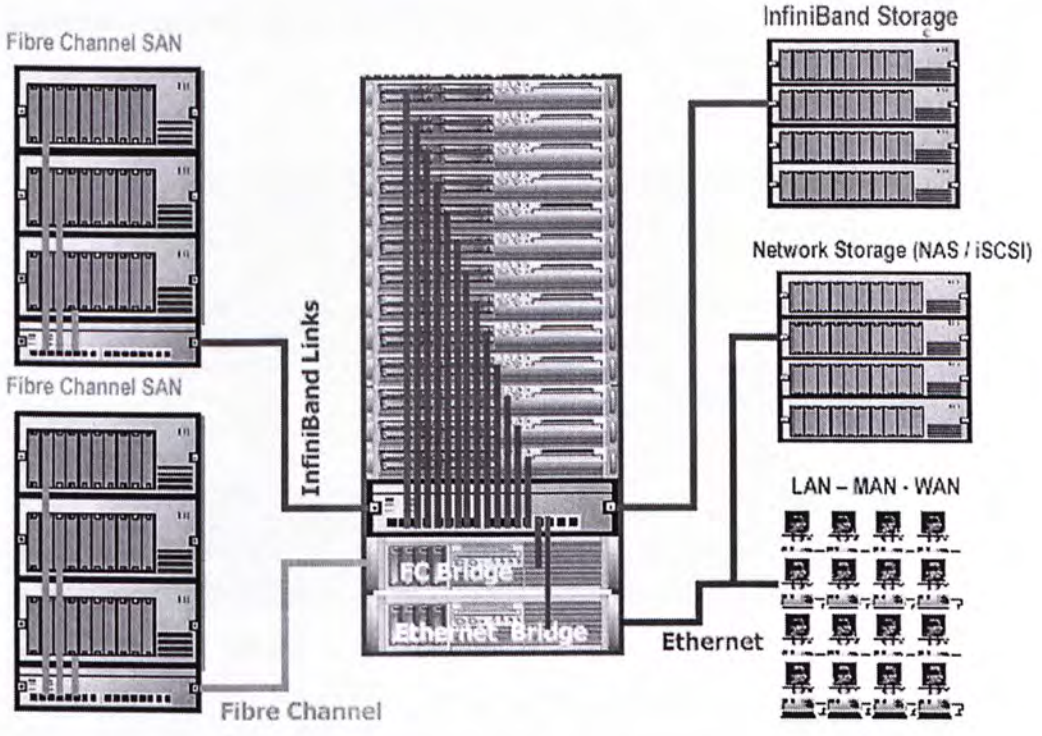


Figure 2.13 Protential Extension of InfiniBand to different storage network protocols

2.2.4 Review on Storage Network Protocol

We have gone through quite a number of storage network protocols in this section. Fibre Channel and InfiniBand are protocols tailor-made for the use of their respective infrastructure, while the others are mainly IP-based.

The three IP storage networking transports are significantly different, but they all provide a common function – transporting block-level storage over an IP network. All three transports enable end users to leverage existing storage devices (e.g. SCSI and Fibre Channel) and networking infrastructure (e.g. Gigabit Ethernet), maximize storage resources to be available to more applications, extend the geographical limitations of DAS and SAN access, use existing storage applications (backup, disaster recovery, and mirroring) without modification and manage IP-based storage networks with existing tools and IT expertise **[13]**.

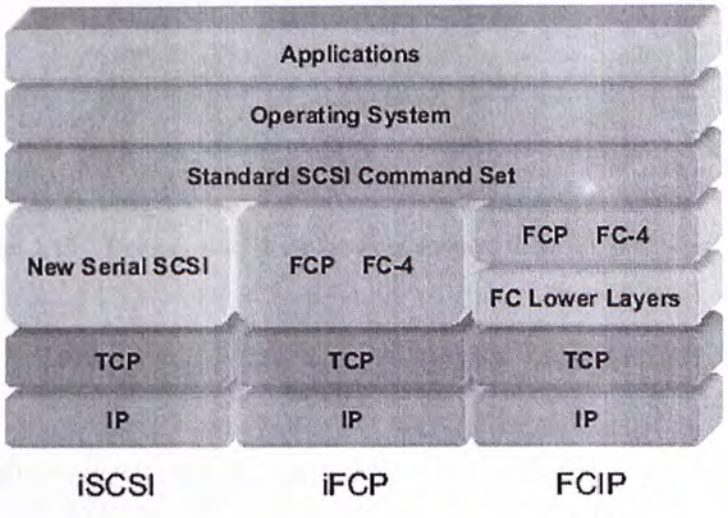


Figure 2.14 Comparing the Protocol Stack of iSCSI, iFCP and FCIP

Despite the similarities and common advantages among these protocols, they actually adopt quite a different approach in details. Figure 2.14 describes the approaches used in each of the protocols.

Figure 2.15 illustrates the protocols supported at each end device and their underlying fabric services. The end device is either a host or a storage device, and the fabric services include routing, device discovery, management, authentication, and inter-switch communication.

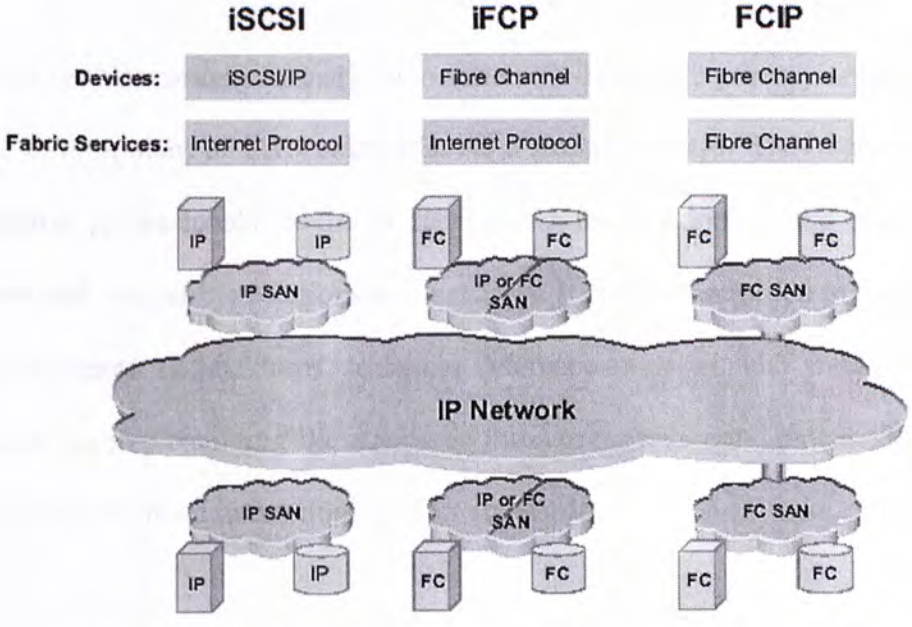


Figure 2.15 Devices and Fabric Services adopted in iSCSI, iFCP and FCIP

IP-based storage network protocols can provide a common way for connecting clients, servers and storage devices, which is built on SCSI and Ethernet technologies. By taking advantages of existing IT knowledge base, these protocols can help increasing the operating distance of storage networks, and lower the burdens of network management by utilizing existing management tools.

Experts at SNIA forecasted the adoption of IP-based storage network protocols will be the mainstream in the coming few years **[14]**.

2.3 Standard Organization

The industry and private networks are yet to realize the enormous potential of SANs. However, a few organizations are actively involved in developing industrial standards and benchmarking the performance of storage network products.

Storage Networking Industry Association (SNIA) is the primary organization for the development of SAN standards. As a forum of major SAN vendors and networking professionals, SNIA is responsible for developing and promoting efficient and compatible solutions in the market. It is also committed to delivering widely accepted architectures, technical reference material, and industry-wide education on implementing the standards through various conferences. SNIA is also actively involved in developing NAS standards.

Fibre Channel Industry Association (FCIA) is an international organization of Fibre Channel product vendors, industry professionals, system integrators, and consumers. It is the combination of two industry associations, Fibre Channel Community (FCC) and Fibre Channel Association (FCA). FCIA focuses on establishing a broad and successful market for the Fibre Channel infrastructure in the field of SAN. It is actively involved in developing a universal infrastructure for Fibre Channel, educating the industry (vendors and consumers) through conferences, and promoting interoperability among various Fibre Channel

products.

Internet Engineering Task Force (IETF) is one of the most important communities of the networking industry and is an active player in the evolution and implementation of formal standards related to SAN management.

American National Standards Institute (ANSI) acts as a middleman between various organizations, rather than actively developing standards. It facilitates the development of SAN-related Fibre Channel standards by establishing consensus among organizations such as SNIA, FCIA, IETF, and so on that are actively involved in the development of SAN standards.

SCSI Trade Association (SCSITA) is an organization of various vendors of SCSI products. It focuses on the promotion of SCSI technology in the field of SANs and is actively involved in developing interoperable SCSI-related standards and educating the market about the advantages of implementing SCSI technology in SANs.

InfiniBand Trade Association (InfiniBandTA) is founded by IBM and currently working on the development of new I/O specifications to facilitate data transfers on switched Fabric technology.

2.4 Summary

In this chapter, various proposals on storage network have been studied. Even thought they apply different technologies in this issue, they are mainly

targeted at enabling high speed, scalable storage network. Actually, no matter the deployment by a special-purpose network or a general-purpose network, storage network can be generalized as a network interconnecting hosts and storage devices.

In conclusion with the variety of proposals on industrial standards, switched-fabric designs have been the mainstream idea for upgrading system throughput. With various proposals in hand, the transmission and switching of storage networks become an important concern in the system performance. We will continue discussing the technologies of switching in storage network in the coming chapters.

3. Switching Design for Storage Networks

In this chapter, we will go into details in the switching aspect of storage networking. As we have concluded in previous chapter, storage networks can be generalized to be a network interconnecting storage and network devices, in regardless of its underlying topology and protocol (as shown in Figure 3.1). The major purpose of storage networks is for cross-connectivity so that information can be shared by different users.

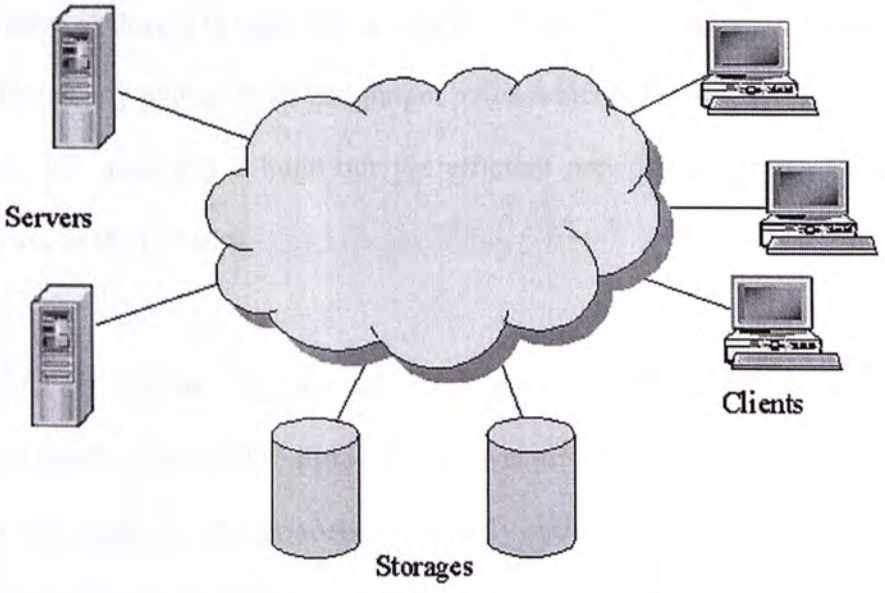


Figure 3.1 Storage network generalized as a network of storage devices and clients.

Storage networking is a broad concept. When we put these designs into production, we have different options in implementing the underlying network.

The easiest and cheapest way is to use a big switch to interconnect all necessary devices together. We will focus our discussion on the simplest case, a single switch. We can easily generalize the problem into any network size by recursive construction.

In networking environment, the two main factors governing the transmission performance are the *transmission bandwidth* and *switching bandwidth*. After the introduction of fiber optical communication, the transmission bandwidth has been improved significantly. However, the switching design is still mainly in electrical domain, which has usually been the bottleneck of the entire system.

In view of the need of speedy communication of storage networks, the underlying switching fabric design has been the crucial part of network performance. A device is qualified as a switch if it can accommodate a connection state between any pair of input and output ports, which gives the ability to process switching job. Building a huge but yet efficient network for storage access is challenging to the industry.

However, the development of switching technologies is far behind the industry's needs. The packet processing power doubles every eighteen months, whereas the doubling of link speed takes only seven months **[15]**. The switches and routes has been the bottleneck of processing in the network, and the situation is getting worse. A faster and more powerful switch is required urgently by the industry.

3.1. Shared Bus Design

In our specification, what we need is a media for interconnecting all network devices together. In order to allow end-to-end connection between any pair of ports, typical interconnection models will be STAR, RING or BUS.

The STAR connection requires a centralized device for information exchange among ports, which is actually a switch. In designing our first switch, we will be focusing on RING and BUS topology. RING and BUS both adopt a common channel so that any client on the media can communicate with any other client, except for their difference in cyclic connection nature. Provided that the interconnection channel has sufficient transmission bandwidth, all the clients or devices can talk with each other, this channel qualifies as a switch.

Figure 3.2 shows a simple design of a network switch, which consists of one central CPU **[16]**. The routing functionalities are mainly software-based. A typical example of this type of switch is a general-purpose computer installed with multiple network interface cards (NICs). Whenever packets from the network come in, the line cards convert the *medium access control* (MAC) protocol into *direct memory access* (DMA) signal, and store the packet content on the memory. The CPU will then load the packet information, deciding where it should be forwarded to, and issue control command to the corresponding egress line card to send out the packet from the memory.

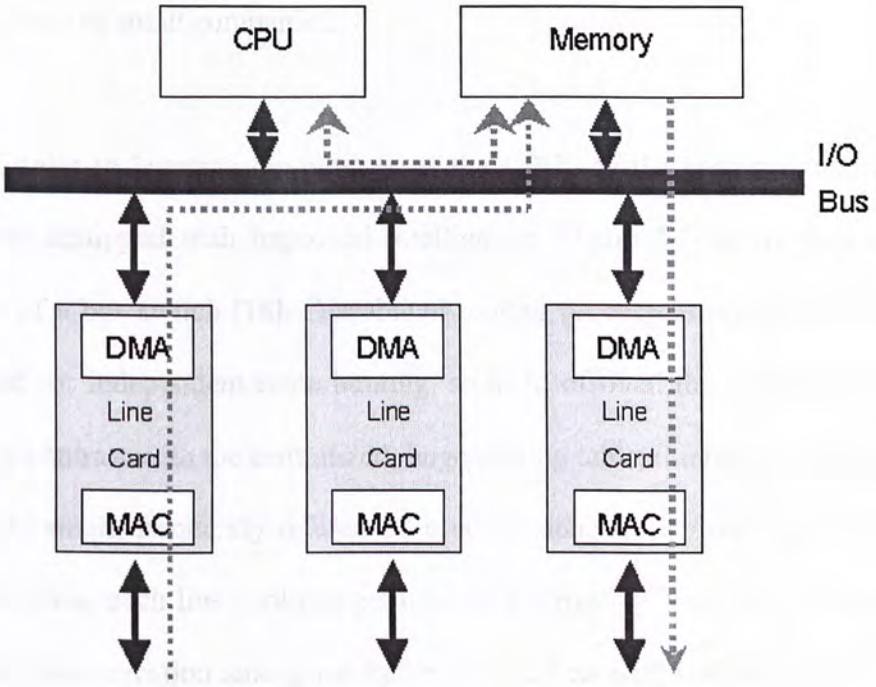


Figure 3.2 A switch using common I/O bus

However, as you may easily notice, the design is not efficient in heavy traffic network. In the process of packet forwarding, a packet needs to travel from the inbound interface to the main memory, and then to the outbound interface. The internal traffic via the common I/O bus is enormous. Moreover, for packet route-hunting process, memory-intensive operations such as table lookups become the major consumers of CPU cycles. Performance of this design is not proportional to the processing power of the CPU, which makes this design not scalable.

We define the efficiency in handling the switching operation as switching bandwidth. The switching bandwidth of this type of switch is limited by the bus speed and the processing power of CPU. This design cannot scale to meet the increasing throughput requirements of the industry, and therefore is only good for

personal use or small companies.

In order to leverage the burdens of the CPU and the common bus, the line cards are equipped with improved intelligence. Figure 3.3 shows an improved version of a bus switch **[16]**. Distributed routing processors are installed at each line card for independent route hunting, so as to off-load the centralized routing CPU. In contrast with the centralized, huge routing table, the line cards maintain a relatively small, frequently referenced routing cache for internal use. With these modifications, each line card can get most of the routing work done. By enabling mutual communication among the line cards, packets with resolved header can be redirected to the outbound interface over the bus once only, and therefore increasing the overall throughput.

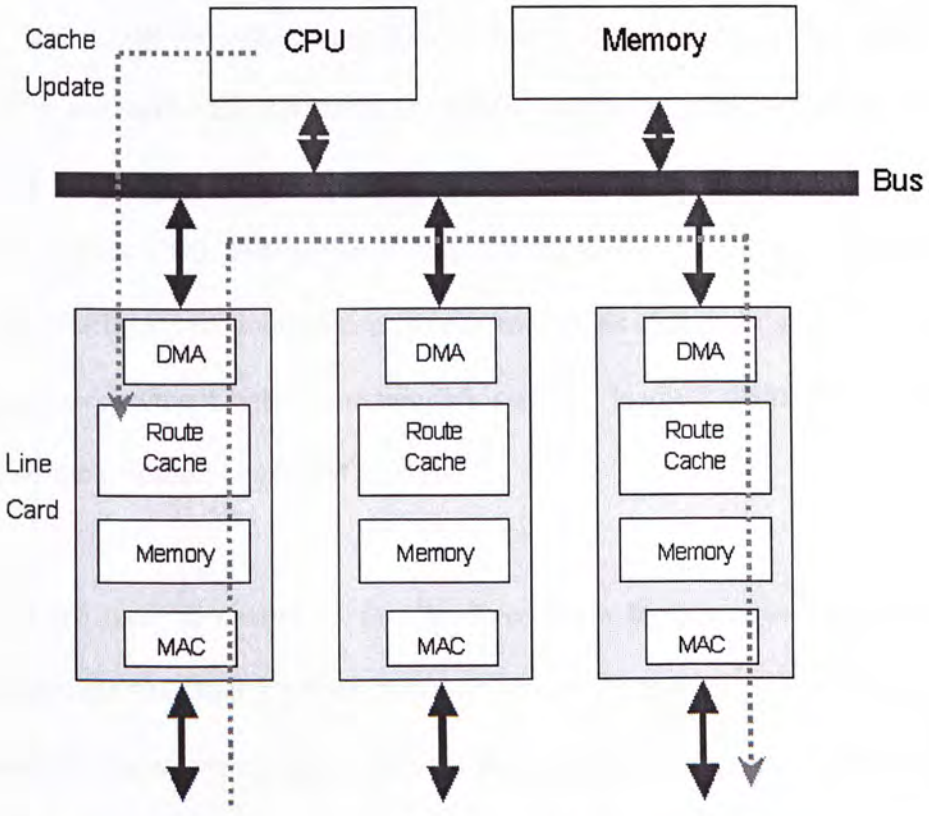


Figure 3.3 An improved version of bus switch with reduced bus transactions

As the route cache of each line card is small, only the most recently referenced information will be stored. Due to the bursty nature of network traffic, this setting can work pretty well. If a route is not available in the cache, the respective packet will be passed to the main CPU for address resolution, as done in the previous version. After the first referencing of the route, this route information will be sent to the inbound interface to update the cache, so that subsequence packets can be served directly. The route information will be periodically aged out so as to keep up with the changing network topology.

A major limitation to this design is that it has a traffic dependent throughput. As the traffic of the network becomes heavy, the route cache will soon become insufficient, and more packets will be passed to the central CPU. The table lookup for a packet can be many times longer than that done by the line cards, and therefore, we can hardly determine the actually throughput of the system. We can improve the performance of this system by adding cache size to each line card, but the shared CPU and common bus can neither scale to high capacity nor provide traffic-pattern-independent throughput. This design is good for storage networks deployment only if the network size and loading is small, or when the speed of the common bus is fast enough.

As we have discussed so far, the three main factors governing switching bandwidth are processing power, memory bandwidth and internal bus bandwidth. In review of the above design, if the shared bus is replaced by other switch fabric, with interconnection unit between interface cards not the bottleneck, the new bottleneck will be packet processing.

3.2. Time Division Switch

The shared bus is the major system bottleneck in the previous setting. However, for a fast enough bus relative to the incoming traffic, the system can actually work fine. The telephony system is a good example of this approach.

For the high-speed network backbone, bandwidth is shared by multiplexing smaller pipes together in time domain. To perform switching on these smaller pipes of signals, switching in time domain is a good option.

Time slot interchanger (TSI) is a device to shuffle the order of the data among time slots. Data are sequentially read into TSI, changing their relative ordering, and are then written off sequentially. When the receiving end reads the data, due to the changed order, data can be sent to different receivers accordingly (as shown in Figure 3.4). By imposing variable delay to each time slot, the content shuffling provides a way in connecting any inbound time slot to any outbound timeslot, and therefore qualifies as a switch.

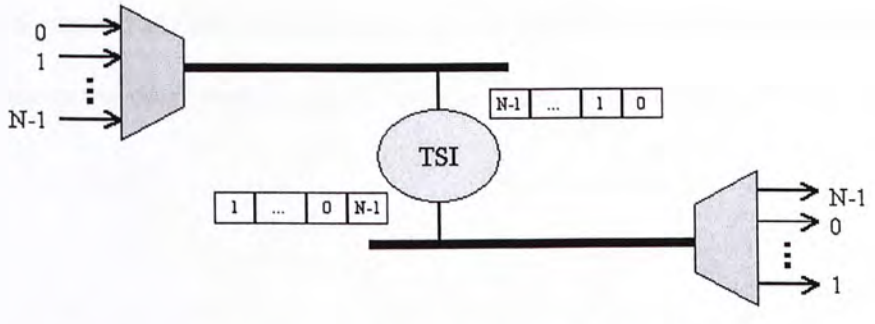


Figure 3.4 Time division switching by means of time sloe interchanger

The design relies the centralized processing at the TSI, which is barely scalable. As the transmission bandwidth and the number of time slots increase, TSI is inefficient in handling the job. Therefore, this switching design is not good for storage networking, where speedy communication and heavy loading are expected.

3.3. Share Buffer Memory Switch

Although the bus switch with common bus and memory is not efficient in handling switching, this is a good start in designing a method to exchange data from input ports and output ports. By integrating with a systematic control mechanism, we can build a *shared buffer memory switch* for the purpose of device interconnection.

Figure 3.5 shows the implementation of a shared buffer memory switch. Share buffer memory switch allows multiple ingress and egress ports in assessing a common pool of memory. Data from an ingress port is temporarily stored in the memory, and is associated with the corresponding output queue. When an egress port reads data from the memory, the data intended for a specific output is then forwarded to the corresponding port, and therefore perform the switching work.

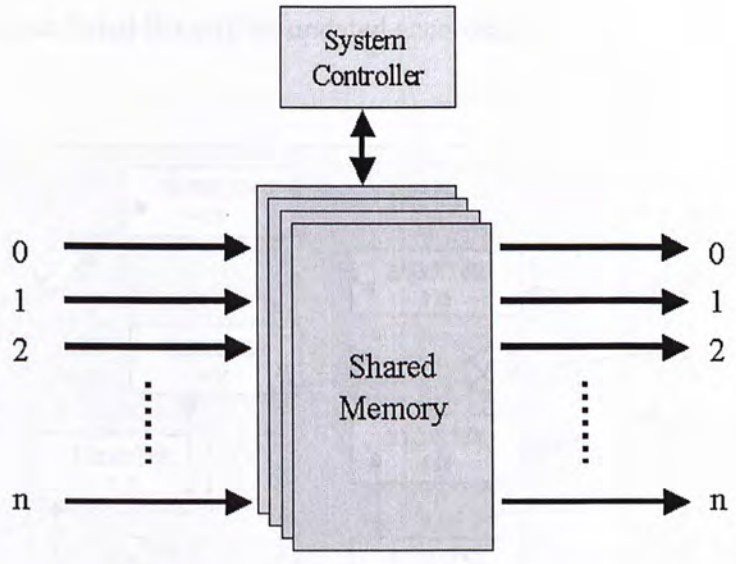


Figure 3.5 Shared buffer memory switch

Share memory switch adopts linked-list as its internal data structure, as shown in Figure 3.6. When data packets come in, the ingress processor will examine the headers of the packets, segment the packet into fixed length cells if necessary, and store them in some common memory. The header information will then be passed to a routing engine for processing. The work of a routing engine is to find out the proper egress port the packet should be forwarded to by means of table lookups, and return the control signal to the queuing controller. As the process of table lookup is slow and repeating, typical switch will distribute this work to an array of routing engine to increase its efficiency.

After the routing engine decides where the cells should be forwarded to, it will append the memory link to the cell to the proper outgoing queue. Whenever an egress port has sufficient bandwidth in handling packet forwarding, the egress controller will visit the corresponding linked list, get the cells from the list, reformat it into packets and send it out. The memory of that cell will then be

released, and the listed list will be updated accordingly.

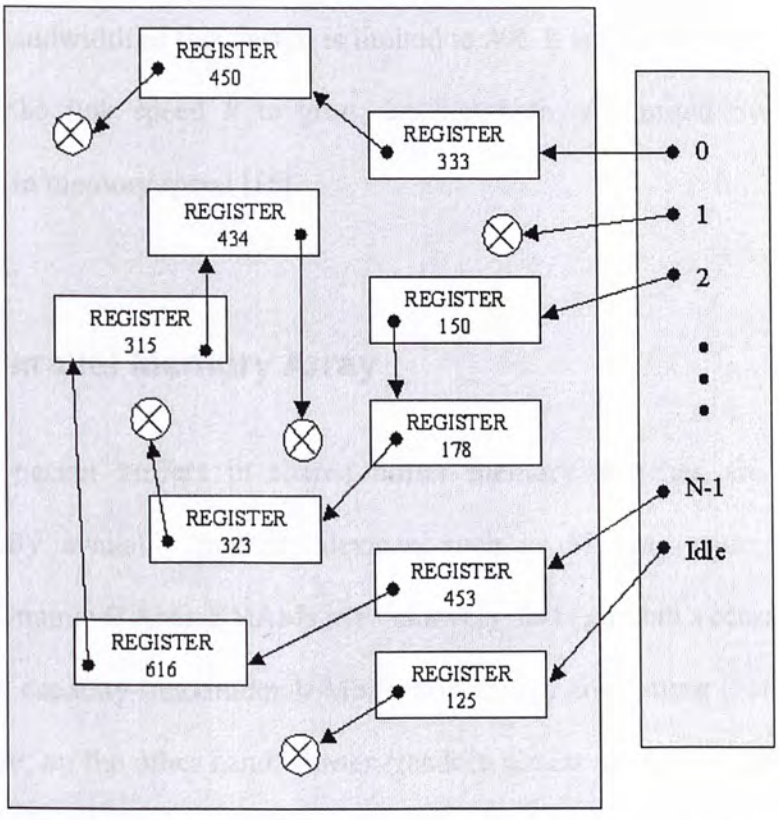


Figure 3.6 Internal data structure of a shared buffer memory switch

Due to the sharing nature of the buffer pool, any output queue can expand length by drawing memory from other queues subject to the overall buffer usage, so as to absorb large bursts directed to any output. This approach can maximize buffering efficiency while minimizing the total memory requirement, and is therefore popular for switch design **[16]**.

However, the I/O latency is a shortcoming of this design. For successful packet forwarding, a packet must first be written to the memory and then read off. Let N be the number of ingress links, each operates at a rate of R. The egress port must be capable in reading a packet in $1/NR$, so that no buffer overflows even in

the extreme case that simultaneous packets come in from all ingress ports to the same egress port. As the system throughput is limited by the memory speed, the switching bandwidth of this design is limited to *NR*. It is fine for either the switch size *N* or the link speed *R* to grow, but not both, as limited by the silicon technology in memory speed **[15]**.

3.3.1 Parallel Memory Array

Most packet buffers in shared buffer memory switches are built from commercially available memory devices, such as SRAM (static RAM) and DRAM (dynamic RAM). SRAMs are relatively fast (random access time 4 ns), but smaller capacity (maximum 16Mbits) and energy consuming (250mW/Mbit). DRAMs are, on the other hand, slower (random access time 40ns), but are larger (maximum 1Gbit) and energy friendly (4mW/Mbit) **[17]**. We would actually be requiring the speed of SRAM, but of the dimensions of DRAM.

Consider a shared buffer memory switch using SRAM with word size of 64 bytes, such that each incoming packet/cell is chopped into a fixed block size of 64 bytes before writing to the memory. Since each packet/cell needs to be written and then read off from the common buffer pool, a minimum of two memory operations per packet is required. Considering tiny packets of size smaller than 64 bytes, the maximum possible throughput of the switch is given by:

$$\eta = \frac{64 * 8bits}{2 * 4ns} = 64\ Gb/s\ .$$

If the switch is connected to Gigabit Ethernet, from the conclusion in previous section, the maximum number of ingress ports is limited to 64; if the switch is

connected to Ten-Gigabit Ethernet, the maximum number of ingress ports is limited to 6. The scalability is limited by the memory assessing speed.

Cascading of SRAM chips can provide a larger size memory, but the memory access speed is not improved. For massive data movement, multiple memory operations are required. Parallel connection of SRAM provides a way to increase both memory size and memory speed.

Incoming packets/cells are chopped into fixed length blocks of a word size, which is 64 bytes in this example. Let m be the number of SRAM chips in the parallel array, and n be the total number of blocks being cut out of a packet. The k^{th} block of packet/cell will be stored in the k^{th} chip of the array under the same memory location as shown in Figure 3.7. Given that total number of blocks n is much larger than the array size m, each SRAM chip will get roughly the same number of blocks. Typical IP packets are of the dimension of kilobytes, an array size of 16 chips can easily keep the assumption valid.

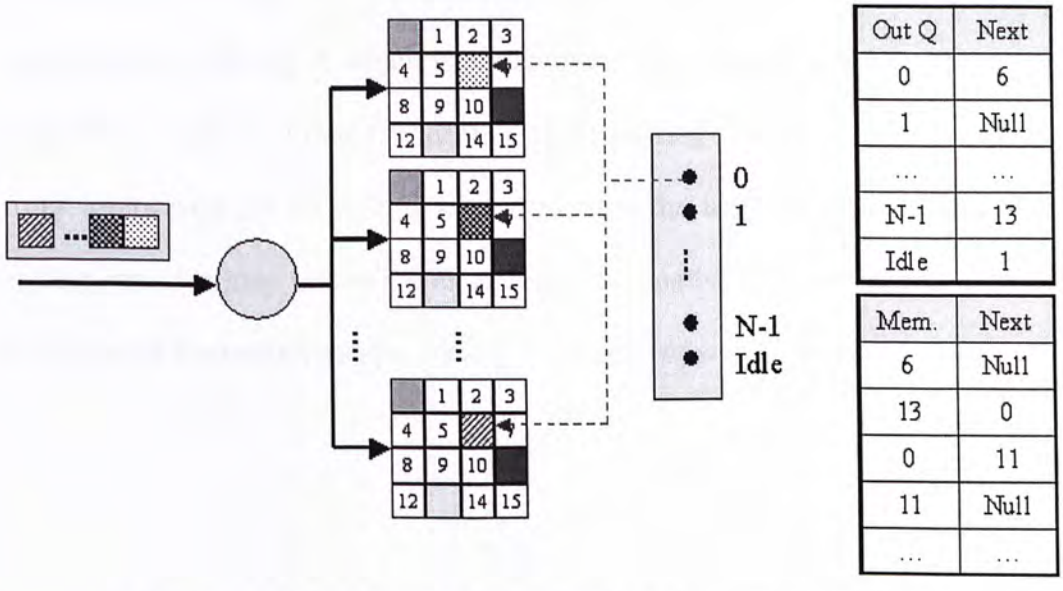


Figure 3.7 Parallel array of memory speed up the memory access speed

When an egress controller decided to dispatch a stored packet, it simply gets the memory address indicated in the output queue and load the memory blocks at each chip. Since the physical assessing time is the lower limit of switching bandwidth, parallel deployment of memory chips can achieve better performance. In the example given in this section, for sequential cascading of SRAM chips, to read 1kbyte data from buffer, we need 16 memory operations, each reading 64 bytes at 4ns, with a total assessing time of 64ns. In parallel deployment, due to the concurrent random access at each chip, we need only 1 memory operation of 4ns for the total of 1kbyte data. Hence, a shared buffer memory switch can reach a higher throughput of

$$\eta = \frac{64 * 8bits * 16chips}{2 * 4ns} = 1\,Tb/s,$$

which hits the typical bandwidth boundary of shared buffer memory switches.

However, a large word size aggregated from a larger array size may not be convenient to use. As in the above setting, the first cell of the packet will always be written to the first chip. For imperfect dividend of cells n by the array size m, fragmentation will occur, which is a wasteful of the memory space in practice. A modified version is to state the identity of the starting chip as an input parameter, from where data can be written to or read from the fragmented free space. This setting can also help to work with tiny packet smaller than mw, where w is the word size of the memory chips, such as ATM cells of size 53 bytes.

3.3.2 Distributive Storage

Whenever packets come into a switch, they are segmented into fixed length cells and are stored in the switching buffer for forwarding purpose. However, as the network size grows, the demand on buffer size increases. Shared buffer memory switch can no longer fit into a single chip. Internal buffering can no longer satisfy the application use, and therefore, the use of external *cell stores* becomes a necessity.

A cell store is a collective system of memory chips for data buffering. The parallel deployment of memory chipsets provides a good way in constructing efficient cell stores. However, due to their off-chip nature, signaling between the switching fabrics is not trivial. Moreover, the buffering efficiency is lowered by the externalized storage. To achieve an approximately the same buffering efficiency, a distributive mix of cell destinations at each cell store is essential.

To raise the overall buffering efficiency, the main concern is to prevent various egress controllers to read from the same cell stores, which produce a bottleneck at the I/O controllers of cell stores and the inter-chipset communication bus. In order to meet the requirement, ingress controllers distribute data cells among data stores in round-robin manner or similar fashion. Assuming cells size is much smaller than the average packet size, each cell store can get approximately the same mix of cell destinations, which collectively achieve a buffering efficiency comparable to a single shared buffer. Figure 3.8 illustrates the effect of distributive storage in I/O bus bandwidth distribution.

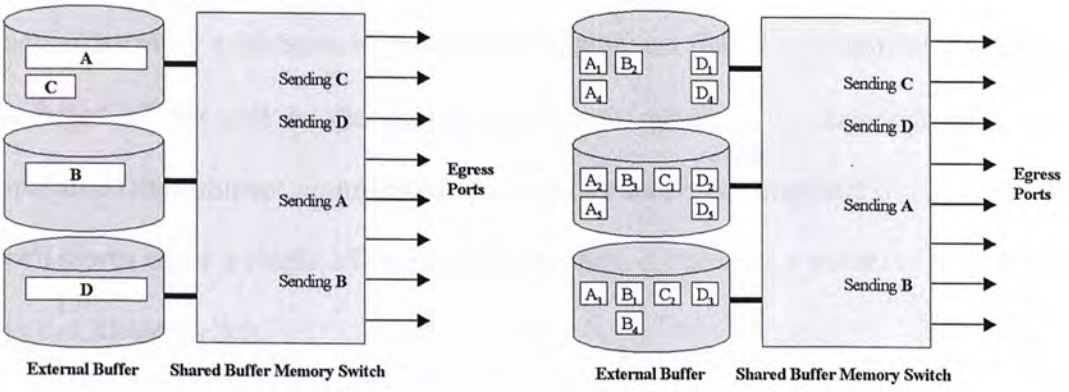


Figure 3.8 Distributive storage leverage the burden of external buffering

For the egress controllers to forward data cells appropriately, ingress controllers require to append cell information to an appropriate output queue. After buffering the payload of a packet in a designated cell store, the ingress controller time stamps the data entry, and passes it together with the routing information in packet header to routing engines for the identification of egress ports.

There are two approaches in handling the payload of packets in egress controlling. In the first approach, the routing engine will append the timestamp of the cells to appropriate output queue, until the concerned egress controller decided to dispatch the cell. The egress controller will approach the appropriated cell store, present the timestamps to the I/O controller, and translate it to the memory addresses of the necessary cells. In this approach, the inter-chipset signaling is simpler, but the address translation table in the I/O controller of cell stores can be large. Therefore, it is suitable only when the latency incurred in output queuing is limited, and hence the size of the address translation table of I/O controller.

On the other hand, the second approach requires the I/O controller to pass the

actual memory addresses to the routing engine, so that it can append the exact location of the cell to the appropriate output queue. This approach relies on massive inter-chipset communication, and is relatively simplistic only when all cell stores share a single I/O controller, but then, it becomes a potential bottleneck in the whole switch.

3.4. Crossbar Switch

Switching is usually done in a shared media, in time domain or in space domain. A crossbar is a space switch composites of a structured array of cross-points, which interconnect the array of input ports and output ports. Figure 3.9 shows a crossbar of size 4x4. All of the crosspoints are controlled by the centralized scheduler so that their states provide proper connection from an input port to the corresponding output port, and hence perform the switching work.

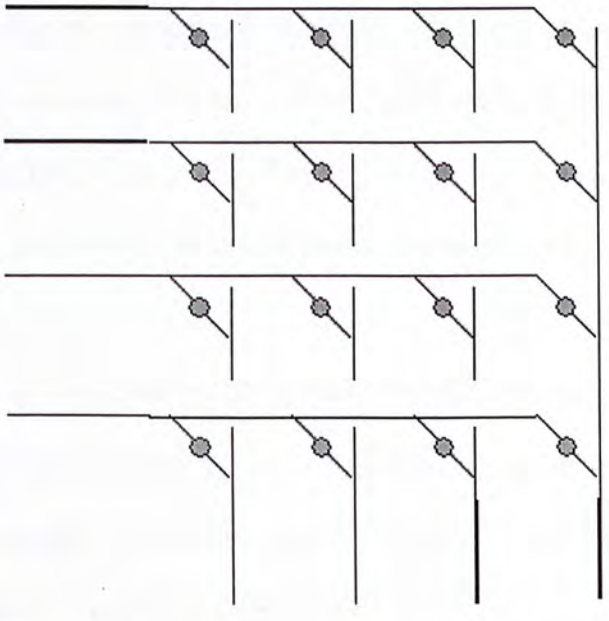


Figure 3.9 A 4x4 crossbar switch.

The non-blocking property of crossbar switch is one of the major attractions to users. Provided that no output contention is present, for an $M \times N$ crossbar switch, we can simultaneous grant k routes between any pairing of k inputs and outputs, where $k \leqq \min\{M,N\}$. Due to the nonblocking property, by proper scheduling of connections, the crossbar switch can work in a very high throughput.

Even though crossbar can provide an efficient on-chip switching, its dimensions are limited due to the difficulties in hardware manipulation and centralized control. When we handle a large number of clients, the use of crossbar become incompatible, as the control signals between chipsets are difficult. Its scalability becomes a great problem.

3.4.1 Arbitrated Crossbar vs. Buffered Crossbar

Since only a single connection is allowed towards any output, there is a need to resolve contention among different inputs. Traditionally input buffer queuing is used to store the data cells temporarily for contention resolution. A crossbar switch under this resolution scheme is called an *arbitrated crossbar switch*.

Recent year, a new variation of crossbar switches has come to the industry. Instead of configuring each crosspoint as an electronic switch, small buffer have been introduced at each crosspoints shown in Figure 3.10. This new design is known as the *buffered crossbar switch*.

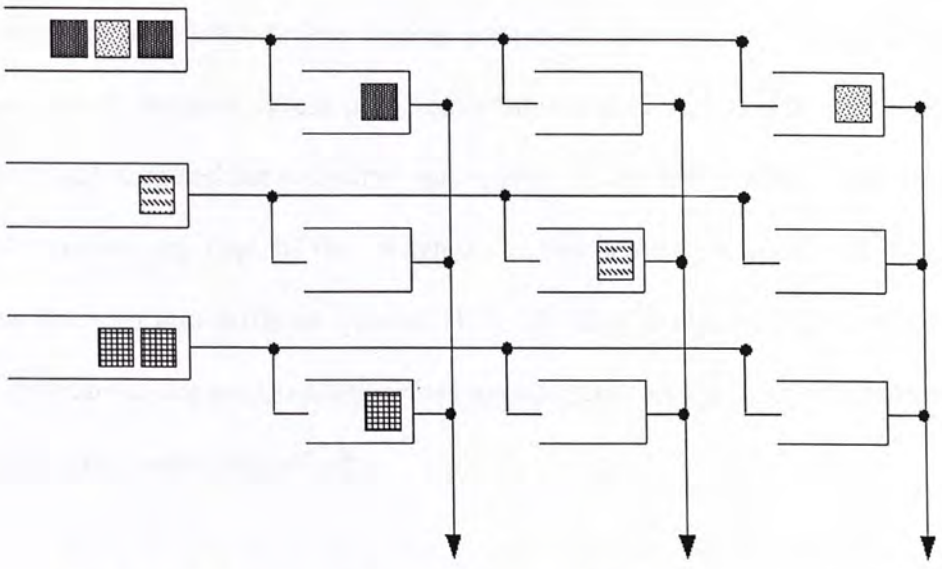


Figure 3.10 Buffered crossbar switch in associated with virtual output queuing

3.4.1.1 Arbitrated Crossbar Switch

In arbitrated crossbar switch, each ingress port contains an input queue. If multiple inputs are targeted to the same output, at most one of them can get forwarded, and the remaining cells will be queued up at their respective input buffer until bandwidth is granted to them.

However, for input queuing with single queue, when the leading packet of a queue is blocked waiting for an available egress timeslot, the late traffic that is targeted at some available egress port may be blocked. The problem is known as head-of-line (HOL) blocking. Due to imperfect utilization of available timeslots, the overall throughput of arbitrated crossbar switch is only 58% for saturated traffic **[16]**. As the queue grows, some data may get lost due to buffer overflow.

The introduction of virtual output queuing (VOQ) can help solving the

problem. For an $M \times N$ crossbar switch, instead of maintaining M universal input queues for all targeted egress ports, each ingress port maintain N virtual output queues, each targeted for a distinct egress port. In another words, a total number of MN queues are kept in the switches. In this setting, as traffic for different egress ports go into different queues, HOL blocking is eliminated. However, the data structure of the switch memory becomes highly complex, and the work of the scheduler also becomes difficult.

Recall the switching bandwidth of shared buffer memory switch, which is its memory assessing speed. Since any output queue has to handle the extreme case of simultaneous incoming packets, it requires a speed of MR, where M is the number of inbound interface, and R is the link rate. In arbitrated crossbar, since contention is already resolved by input buffers, the crossbar fabric can provide its service at the link rate R, where scaling up is considerably easier.

3.4.1.2 Buffered Crossbar Switch

In arbitrated crossbar switches, input queuing is employed. With VOQ working at each ingress port, the dimension of ingress-to-egress matching is huge, which makes pre-forwarding scheduling a difficult task. Scheduling methods supporting quality of service (QoS), such as weighted round robin (WRR) or sometimes known as weighted fair queuing (WFQ) become not trivial, as multiple connections for different egress ports are prohibited from the same ingress port.

The solution commonly used today is to provide significant internal speedup, where the crossbar port rate is higher than link rate by a factor of f. Since the

average utilization of the switch is only $1/f$, imperfect matching in scheduling can also be acceptable. Moreover, since queues are now tended to build up on the output side, a combined input-output scheduling makes it possible to maintain QoS at output side.

With the buffer of the new design, the head-of-line cells for each egress port are pre-sent to the buffer at the crosspoint, which is directly connected to the egress port as in Figure 3.10. Once there is sufficient bandwidth at an egress port, we can handle cell forwarding from any ingress port as the single-connection constraint is leveraged. Scheduling is dramatically simplified and weighted round robin (WRR) becomes feasible, no internal speedup is needed, and, in many cases, no output buffer memories are needed. This brings important advantages to crossbar switching **[18]**.

3.4.2 Switch Scheduling

In input queuing switches, such as crossbar switch, connections can only be granted when there is no output contention. To guarantee collision-free routing, a pre-forward path matching has to be done, known as scheduling. Scheduling is typically done in the scheduler of a switch, where centralized control intelligences are present. The major task of a scheduler is to calculate feasible path allocation from ingress ports to associated egress ports in a timeslot, and issue appropriate control signals to the switching fabric for proper forwarding of network data.

3.4.2.1 Bipartite Matching

Network specialists have proposed the use of *bipartite matching* as the scheduling method **[19]**, which is a method in finding a (maximum) pairing between two disjoint sets of vertices. An example of bipartite graph matching is shown in Figure 3.11.

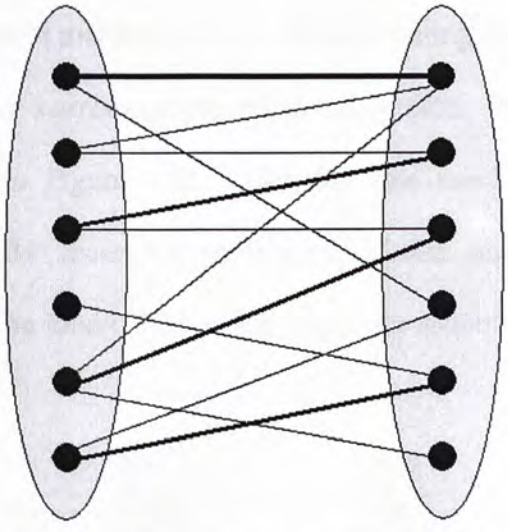


Figure 3.11 Application of bipartite matching in switch scheduling

We can use the *augmenting path algorithm* in finding the maximum bipartite matching **[20]**. Starting from any feasible matching solution, where an empty matching is typically used for initialization. For any unmatched vertex in the left set, we perform a depth-first search-like evaluation to consider the revision of current matching. The traversing will go from left to right through an unmatched edge, known as *red edge*, and from right to left through a matched edge, known as *blue edge*. The traversing will continue until there are no suitable edges to go. If it stops at an unmatched right node, the path is a feasible modification on the existing matching, and is known as an augmenting path.

Note that for any possible augmenting path, it starts from a vertex of the left set and ends at a vertex from the right set. The number of red edges is always one more than that of the blue edges. By accepting the suggested amendment, the resulting bipartite matching is always larger in size by one. After the discovering and revising of all augmenting paths for each left vertices, the resulting bipartite matching is maximal. Taking as an example the bipartite matching in Figure 3.11, which is not a maximum matching. Hence an augmenting path exists. By revising the matching by the corresponding augmenting path, the resulting bipartite matching is shown in Figure 3.12, which has one more connection than the previous example. By recursive application of the algorithm, if no more augmenting path can be found, the resulting bipartite matching is maximal.

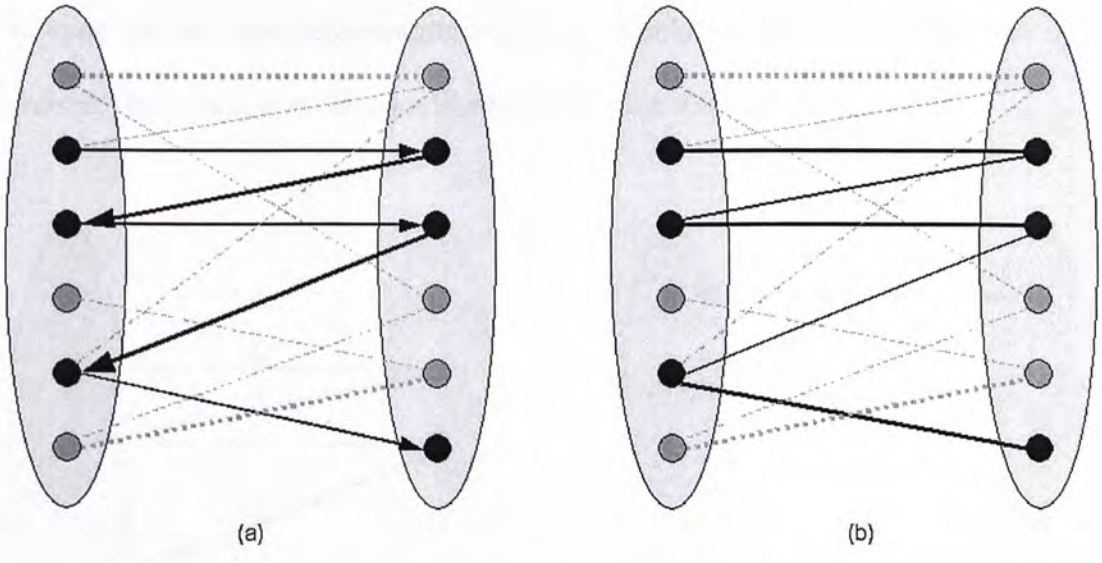


Figure 3.12 (a) The discovering of an augmenting path (b) Revising of bipartite matching with the discovered augmenting path

The bipartite matching is basically dealing with unweighted graphs, which correspond to unweighted network traffic. To include the weighted case, a slightly

modified algorithm can be used instead. Note that the idea of an augmenting path is a trial reconfiguration of the current matching. If the weights are considered in the searching of augmenting path, the resulting matching will also be maximal in terms of total weight.

In addition to the searching of augmenting path, we also consider its feasibility. While traversing on the edges, we will at the same time calculate its fitness function value. The fitness value will be initialized to be zero, and the edge weight is added to it while traversing a red edge, and is subtracted from it while traversing a blue edge. Upon successful discovery of augmenting path, the resulting fitness value tells us the net gain in total weight of the resulting matching. For positive fitness path, we define it as *feasible augmenting path*; otherwise, we define it as *infeasible augmenting path* (as shown in Figure 3.13). In updating our weighted bipartite matching, if only feasible augmenting path is chosen, the resulting matching will be optimized in total weight.

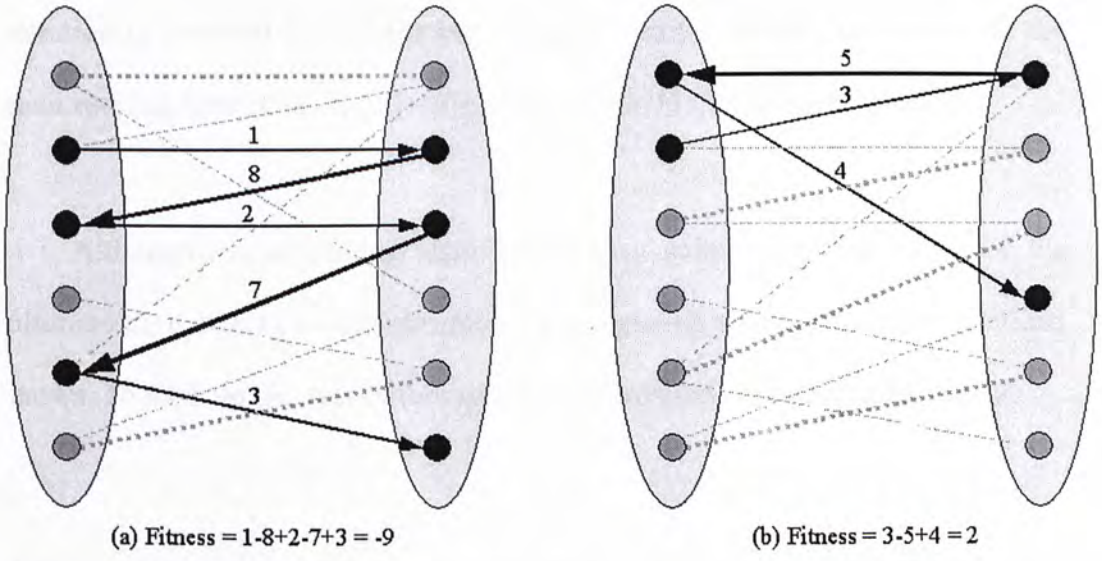


Figure 3.13 (a) An infeasible augmenting path (b) A feasible augmenting path

Scheduling by bipartite matching attempted to give a maximal matching, which make the best utilization of the network bandwidth. However, the algorithm does not take the time delay a packet experienced into consideration. Starvation may occur, which severely degrade the service. With the weighted bipartite matching, ageing can be used to solve the problem. Whenever a packet in the switch has been buffered waiting for a certain period of time, its priority will be raised by the switch to speed up its processing time. Similar strategy has also been adopted in CPU scheduling in computers' system software. Further discussion on this issue will be given in Chapter 4.

The maximum bipartite matching is a good model for network traffic scheduling in input queuing switch. However, the algorithm running time is huge. The algorithm targets at finding an augmenting path, which has its size bounded by $O(|V|+|E|)$, where V is the set of all vertices and E is the set of all edges. The algorithm is iterated until a maximum bipartite matching is found. The number of iteration is bounded by the number of vertices in the left set, and therefore the total running time is $O(|V|\cdot(|V|+|E|))$, which is $O(|V|\cdot|E|)$ for most cases.

Although the scheduling algorithm runs in polynomial time, it is still the ultimate limitation in switch scheduling and signaling when the number of clients grows. To push up the overall throughput, a distributed algorithm is necessary.

3.4.2.2 Token-based Distributive Scheduling

In order to eliminate the bottleneck of centralized control, a distributed

algorithm has been proposed in this section, which makes use of the concept of token. A distributed algorithm requires the participants to obey a certain set of self-governing rules, so that centralized control can be eliminated.

In input queuing switches, an egress port is in some sense "shared" among all ingress ports. They have to get the permission from the scheduler before sending data. At each time slot, only one ingress port can be granted with transmission permission. In another word, multiple ingress ports are fighting for a single egress port at a time slot.

We will be introducing the use of token in switch scheduling. Since there can only be single connection to a particular output, one token is associated with each egress port, where all tokens are maintained by the scheduler. An ingress controller can send packets to an egress port only if it gets the token for that target port. Except for the case of multicasting, each ingress controller can only hold one token at a time for the fairness of the algorithm. The scheduling mechanism is used in associated with VOQ to prevent HOL blocking.

Round-Robin Polling

One of the approaches is to let the scheduler to poll each ingress controller for transmission request in round-robin manner. Similar to that done in token ring, an ingress controller intended to send packets to a specified output port will get the respective token passing by, and start sending data in the coming time slot. After the transmission, the token will be released to the scheduler, and the scheduler will start polling the following ingress controller.

This approach is, however, inefficient. The scheduler will be performing the polling task repeatedly, but most of the time, the polling is redundant due to idle port or token mismatch. Moreover, this approach is not quite fair in scheduling. Consider the i^{th} ingress controller is intended to send packet to the j^{th} output port. Unfortunately, the token has just passed and the scheduler is polling to the $(i+1)^{th}$ ingress controller. The packet has to wait for a whole cycle before the token for the j^{th} output port comes back. On top of that, if a new packet intended to go to the j^{th} output port comes to the $(i-1)^{th}$ input port, the previous packet will need to wait some more timeslots before it can actually be served, which is unfair to the early-comers.

Token Knock-out Competition

To speed up the searching of a potential candidate, a binary search tree or similar algorithm can be used. Instead of polling in round-robin manner, the scheduler can initiate a token competition among ingress ports. A group size of two ingress ports can compete for a token if binary search tree is used. The request weight can be a good measuring quantity for the competition. The winning one will be advanced into the next round of competition, and finally the ultimate winner can get the token and forward the packet. The runtime of this approach is in $O(\log n)$, which is better than $O(n)$ of the round-robin fashion.

However, there is a potential problem when an ingress port gets more than one token. This is not a problem if buffered crossbar is used, as the head-of-line packets are already buffered at the crosspoint. It can be a problem if arbitrated crossbar switch is used. An ingress port cannot send packets to different egress ports simultaneously, implying a potential waste of some timeslots.

On-demand Token Distribution

To leverage the redundant workload of the scheduler, a modified approach is proposed. In this setting, all the tokens are still maintained by the scheduler, but the scheduler is only required to provide service upon request from ingress controllers. To get the permission of transmission, the ingress controller has to request for the token of the corresponding egress port directly from the scheduler. Unnecessary processing time is saved, and also, as the token requests are real-time, unfairness has been improved.

However, problems may occur when an ingress controller cannot get a free token. Depending on implementation details, the ingress controller may keep on requesting or sleep until the scheduler inform for an available token. The "busy-waiting" hypothesis may occur. Besides, due to the independent bidding of tokens, an ingress controller may again get more than one token at a particular time. This strategy is best used in buffered crossbar switch. For the deployment in arbitrated crossbar switch, intelligence should be made to the ingress controller to send negative acknowledgement to the scheduler, freeing the excessive token for other controllers without inferring its original queuing position.

3.4.2.3 Resource Counting using Semaphore

The design of the token-based scheduling method provides a good way in distributing scheduling workload to ingress controllers, but it requires intelligences at the controller in re-attempting on request of token, or to signal any sleeping controller to proceed. One of the implementations on related issue in

computer technology is by using *semaphore*. A semaphore is basically an atomic counter for controlling the access of certain protected resources.

In this section, we will be presenting another scheduling method by means of semaphore. As we have discussed, semaphore is an atomic counter. In the application in critical resource access control, each attempter has to get a count from the semaphore, and the semaphore will decrement by one. Successful attempter has to obtain a positive count before proceeding, while unsuccessful candidates will be waiting for signal to proceed. In this new approach, we will be applying the use of semaphore in another area, resource counting, into scheduling design.

We will be simulating the effect of output queuing in input queuing switches by means of semaphore. As in the previous designs, each egress port is associated with a semaphore, and all semaphores are maintained in the scheduler. To simulate output queuing, a queue in associate with an egress port is necessary. The semaphore for each egress port provides this function. The semaphore distributes the queuing matter into different ingress controllers, letting them know where their queuing position is. When an ingress controller gets to the head of queue, it is permitted to forward data.

The implementation of this design is simple. In this approach, the semaphores are initialized to be zero. Whenever an ingress controller request for transmission, it has to approach the scheduler and seek for queuing information from the corresponding semaphore. The information of the semaphore is copied to the ingress controller, and the semaphore will increment by one. In this way, the

semaphore can always provide up-to-date queuing information to ingress controllers. At each timeslot, only those with queuing position zero have the privilege in sending data. Upon each successful timeslot, all semaphores in the scheduler as well as the information obtained by the ingress controllers will automatically decrement by one, so as to reflect the fact that the head of each queue has been discharged. This method actually constructs a distributed VOQ for each output port. Figure 3.14 shows a scenario for an ingress controller to get the corresponding egress queuing position from the semaphore.

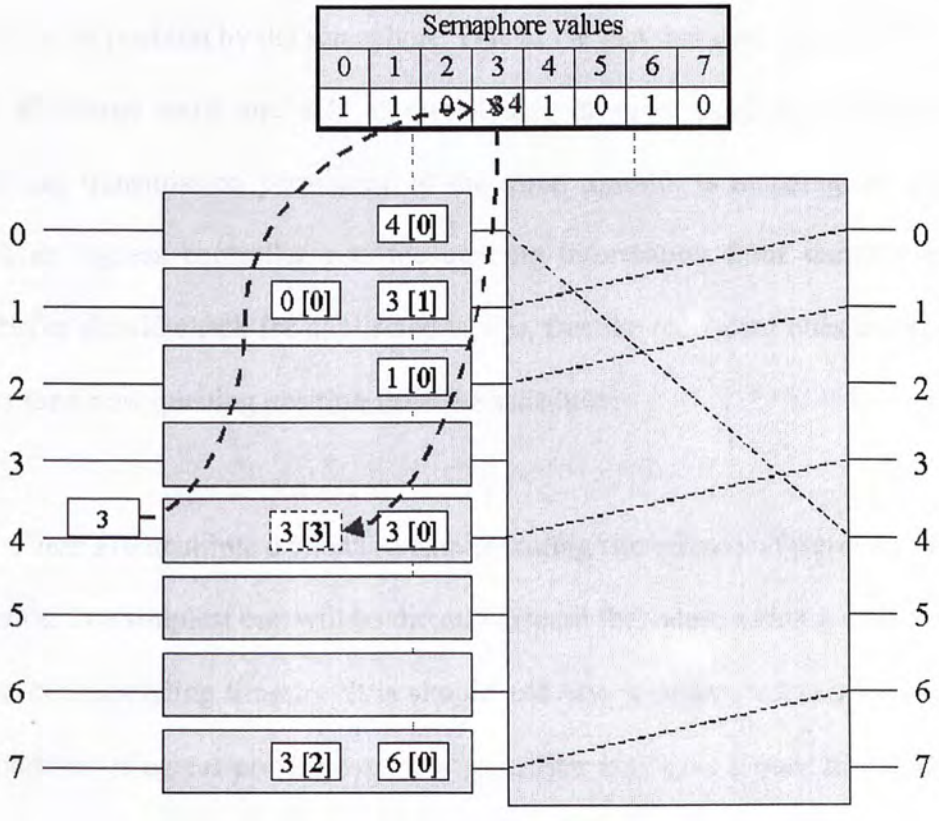


Figure 3.14 Self-service queuing by resource counting with semaphores

The semaphore of this approach gives a good insight to the ingress controllers about the collective queuing position of the "output queue", so as to estimate approximate waiting time for each data cell. This approach works like an

output queuing approach, which gives qualitative service in nature. However, since the cells are physically buffered at input side, even if the egress port is available, data may still not be forwarded as the ingress link can be occupied by another cell. Hence, this approach applies preferably to buffered crossbar switch. In arbitrated crossbar switch, the queuing information stated in the semaphore may not reflect the exact waiting timeslot required.

To solve the contention of entering arbitrated crossbar switch (or other input queuing switch) at the ingress port, additional work has to be done in assigning the queuing position by the semaphore. Due to the fact that each ingress controller may discharge only one cell to output side at each timeslot, multiple cells obtaining transmission permission at the same timeslot is meaningless. Hence, when an ingress controller gets the queuing information from semaphore, the controller should check for duplicated values, free the redundant ones and request again for a new queuing position from the scheduler.

There are multiple options in implementing the release of permission of a timeslot. The simplest one will be directly discard the value, which means a waste to the corresponding timeslot. It is simple and also accurate, but in a long run, if the number of egress ports grows, this algorithm may give a poor throughput. A more realistic solution is to adopt a free list, which holds the timeslots that are released by ingress controllers. In this setting, ingress controllers will first check for vacancies in the free list before getting information from the semaphore. This free list is meaningful only if there is another incoming packet after a release of transmission privilege, of which the timeslot has not expired. It can help pushing up the throughput of the system when traffic is heavy.

With a distributed algorithm in switch scheduling, centralized control is no longer necessary. The bottleneck of switching bandwidth is no longer the centralized signaling processor, but the packet processing time and the underlying transmission bandwidth in each interconnection line. The performance in the next generation switches will definitely be improved.

3.5. Algebraic Switches

Algebraic Switching Theory is found by Li **[21]**. Theoretic concepts in it lead to a family of switches with special properties, which are preserved under recursive construction. By making use of their special properties, we can construct useful switches for use in real life.

By an *algebraic switch* we mean any switching design based on algebraic switching theory. Algebraic switch is actually a technique in building a switch, and its underlying media can vary. If we use copper cable for the core fabric, then the switching bandwidth will be limited to the capability of the cable. If we need a faster switch, all we need to do is to replace the connection inside the switch with faster media, such as optical fibers.

Below we introduce three examples of algebraic switches, which are based on *conditionally nonblocking properties*, *self-routing property* and *multi-stage interconnection of self-routing concentrators* of algebraic switching theory.

3.5.1 Switching by Conditionally Nonblocking Properties

One of the important features of algebraic switches is the preservation of *conditionally nonblocking switching properties*. From the system point of view, if we can properly impose certain kind of pre-switching operation such that they satisfy some specific conditions, the switch can guarantee that the requested connection state can always be granted, that means no blocking will occur during the switching process. Of course, when we are talking about nonblocking, the first and ultimate condition is the absence of output contention, i.e. the competition of the same output port by different ingress packets or data cells. To resolve output contention, input buffering can be used prior to the entering of the switch, as done in crossbar switch.

In algebraic switching, switches are built recursively by smaller switching components in an interconnection network. This network is a routable network, where a route is present from any input port to any output port. The switch realization by recursive construction from smaller switches is called a *switching network*, as shown in Figure 3.15.

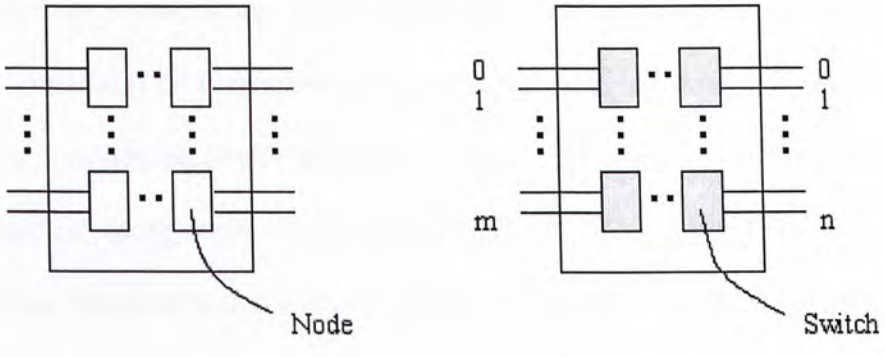


Figure 3.15 A routable interconnection network and a switching network

In building a sizable switch, we need to deal with two independent issues, the switching at each smaller switch and the interconnection among these switches. On-chip switches, such as 2-state switches, deal with the first issue. The second issue is mainly dealt with by studying interconnection networks. The nodes of such networks are usually grouped into stages for easy management and performance investigation.

In particular, multistage interconnection networks that have exactly two stages are known as *2-stage network*. Due to its compact and intuitive structure, we can use it as the basic building blocks for sizable switches. 2-stage interconnection network is unique-routing, so that the control and signaling will be easier than alternative routing network. Unfortunately, 2-stage networks are blocking due to its unique-routing nature. Consider two packets coming from different input ports of the same node, and are targeted to different output port of the same node. Despite the absence of output contention, we can see that concurrent connection can never be granted.

Even thought 2-stage interconnection network is blocking, it is "conditionally nonblocking" in the sense that every combination of external I/O pairs is routable if no two routes are from the same input node to the same output node. By applying particular transformations to the input or output addresses, we can construct some other conditionally nonblocking switches with more useful properties. The adoption of input exchange and output exchange are viable ways in fine-toning switching properties.

The *2X version of 2-stage interconnection* is defined by the 2-stage interconnection network prepended with the inverse of the inter-stage exchange, known as an *input exchange*. The *X2 version of 2-stage interconnection* is the 2-stage network appended with the inverse of inter-stage exchange, known as an *output exchange*. Figure 3.16 shows the 2X and X2 version of 2-stage interconnection network respectively.

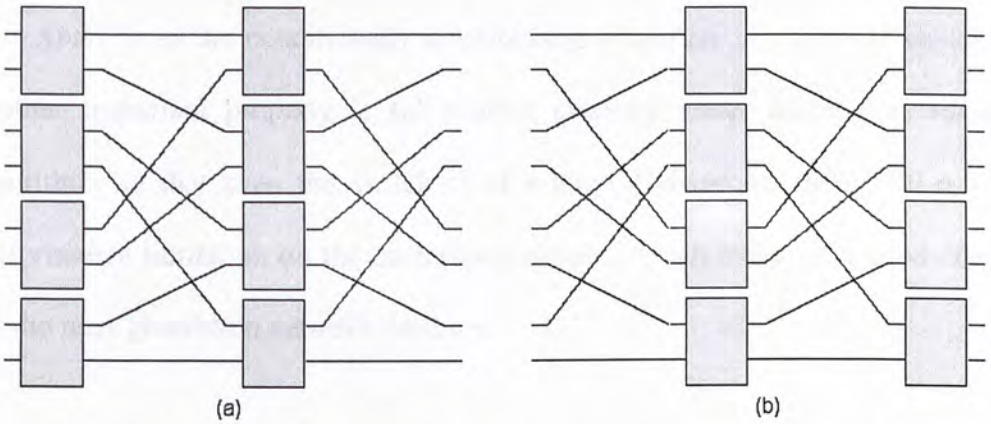


Figure 3.16 The 2X and X2 versions of 2-stage interconnection network

With this new generation of switching networks, we will go ahead and investigate the conditionally nonblocking properties they preserve. Note that an input exchange (resp. output exchange) re-organizes the ordering of input addresses (resp. output addresses), the conditionally nonblocking of 2-stage interconnection network has been changed accordingly. In this new setting, consecutive input and output connections become possible.

In-depth investigations on these switching properties have been made, and it is discovered that specific switching properties are preserved under recursive 2X and X2 constructions. The *compressor property* of a switch is preserved by 2X

interconnection; while the *decompressor* and *expander properties* of a switch are preserved by X2 interconnection **[21]**. These results give a qualitative start in constructing an algebraic switch, where useful properties can be extended to any larger size of switch.

3.5.2 Self-Routing Mechanism with Zero-Bit Buffering

Apart from the conditionally nonblocking properties of algebraic switches, another important property is self-routing property under distributive routing algorithm, so that even the switch is of a huge dimensions, there will not be performance limitation on the centralized control, which makes it a good choice for the next generation network switches.

Recall that the nodes in multi-stage interconnection networks are in arbitrary size. A special family of network, where each node is of the dimensions of 2x2 and each interstage exchange is bit permuting, is called *bit permuting networks*. Moreover, a $2^n x 2^n$ n-stage, routable bit-permuting network is called a *banyan-type network*. In particular, Figure 3.17 shows a 16x16 banyan-type network.

As described before, a KxK exchange is a permutation from 0 to K-1. A $2^n x 2^n$ bit permuting exchange is an exchange induced by a permutation on integers from 1 to n. It gives a permutation on the bit pattern on the induced address at each node. A $2^n x 2^n$ exchange induced by a bit permutation σ is denoted by X_σ.

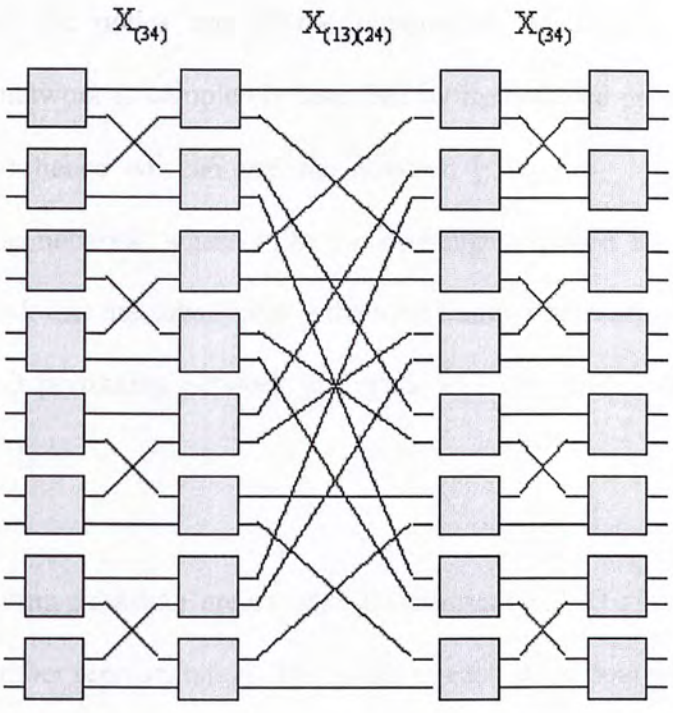


Figure 3.17 A 16x16 banyan-type network

Figure 3.18 is an example of a permutation, where the number represents the bit number of an address. Note that a linear addressing scheme is applied, with the leftmost bit being the first bit. We can use "cycle representation" in group theory to represent a bit permutation. When numbers in a cycle are rotated, the permutation remains the same. As in the following example, $\sigma = (1435) = (5143) = (3514) = (4351)$ and $\pi = (34) = (43)$.

	1	2	3	4	5	6	7	
σ	↓	↓	↓	↓	↓	↓	↓	σ: 1⇒4⇒3⇒5⇒1
	4	2	5	3	1	6	7	
π	↓	↓	↓	↓	↓	↓	↓	π: 3⇒4⇒3
	3	2	5	4	1	6	7	

Figure 3.18 An example on bit permutation

As all of the nodes are of the dimensions of 2x2, a $2^n x 2^n$ k-stage bit-permuting network is completely specified by the induced permutations of its exchanges, and hence we can use the notation $[: \sigma_1 : \sigma_2 : \ldots : \sigma_{k-1} :]_n$ in representing the network, where σ_j is the exchange between the j^{th} and $(j+1)^{th}$ stages for $1 \leqq j < k$ and the subscript n is the total number of addressing bits. As an example, the bit permuting network in Figure 3.17 can be denoted by $[: (34) : (13)(24) : (34) :]_4$.

Bit permuting networks are of special importance due to its close relation with binary number representation. The nodes at each stage deal with the issue of bit conversion, while the bit permuting network deals with bit interchanging between each stage. By applying proper control, packets can be self-routed through the network.

Before continuing to the self-routing properties of bit permuting networks, we will introduce an important tool, the *trace transform* and the *guide transform*. The trace transform (or simply *trace*) of the k-stage network $[: \sigma_1 : \sigma_2 : \ldots : \sigma_{k-1} :]_n$ means the sequence

$$n, \sigma_1^{-1}(n), (\sigma_1\sigma_2)^{-1}(n), \ldots, (\sigma_1\sigma_2\ldots\sigma_{k-2})^{-1}(n), (\sigma_1\sigma_2\ldots\sigma_{k-1})^{-1}(n),$$

and the guide transform (or simply *guide*) means the sequence

$$(\sigma_1\sigma_2\ldots\sigma_{k-1})(n), (\sigma_2\ldots\sigma_{k-1})(n), \ldots, (\sigma_{k-2}\sigma_{k-1})(n), \sigma_{k-1}(n), n$$

We will illustrate this concept by studying an example. Consider the network shown in Figure 3.19, which can be represented as [: (34) : (14) : (24) :]. The trace and guide can be found by bit-by-bit study:

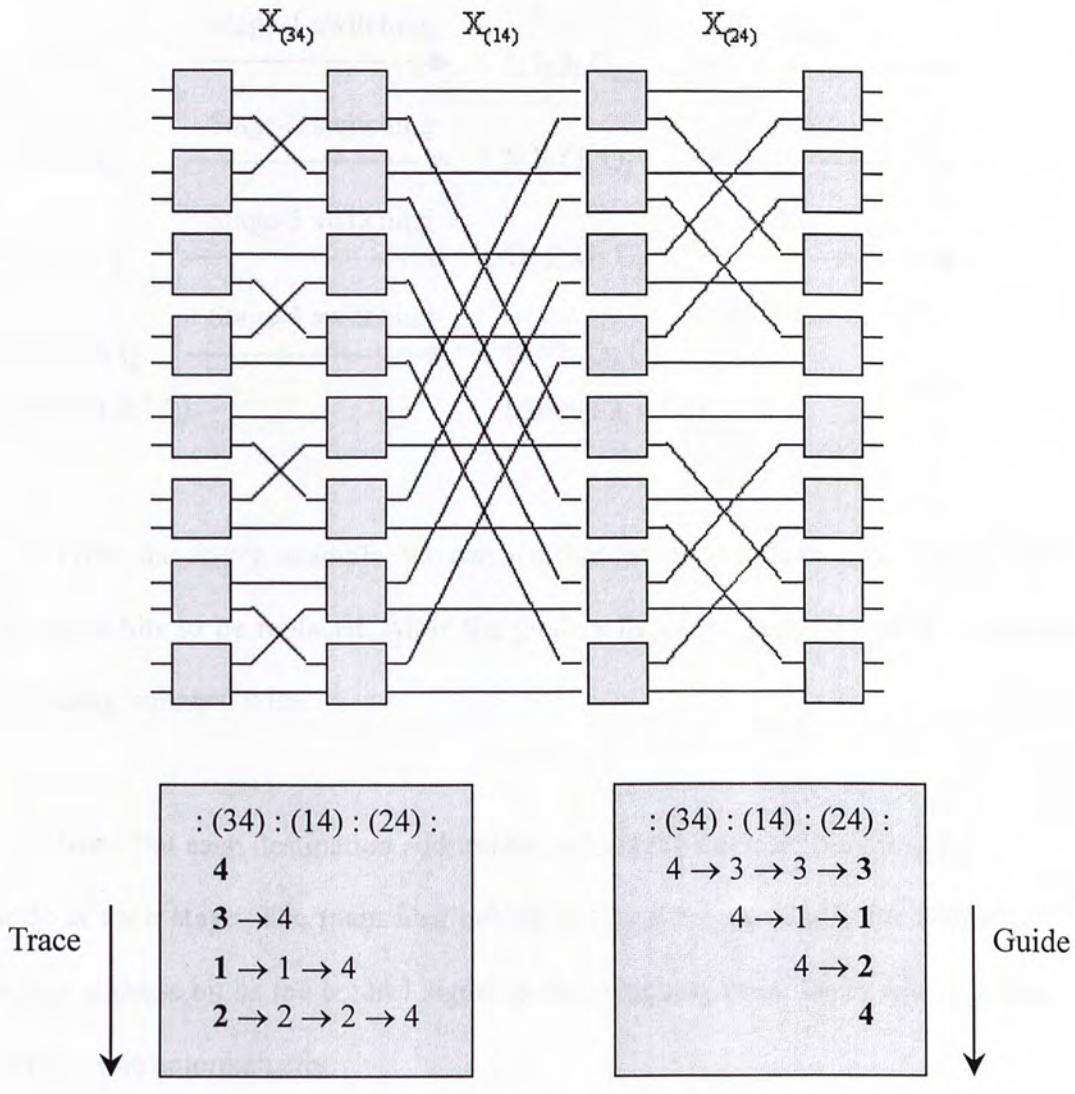
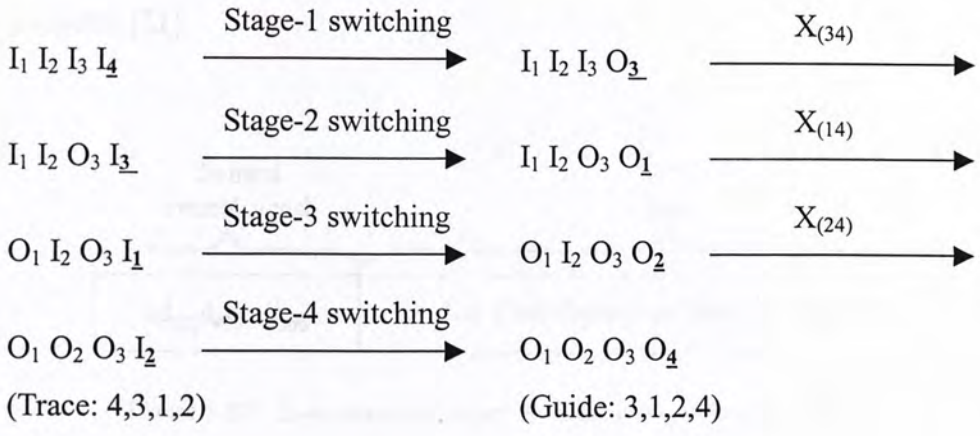


Figure 3.19 Trace and guide calculation of bit permuting network

The trace and guide provide important information of the bit-permuting network. Recall that nodes of bit permuting networks are dealing with the bit conversion at each stage. Let's revisit the network [: (34) : (14) : (24) :] in Figure 3.19, the trace and guide are 4,3,1,2 and 3,1,2,4 respectively, as found above. The stage-by-stage I/O address progress induced by the exchanges of this network is as followed:



From the above example, we can see that the trace tells us the sequence of the input bits to be replaced, while the guide tells us the sequence of the output bits being replaced with.

Note that each destination address bit reflects the intended outgoing port of a node at each stage. The main idea behind is that if we can apply the respective output address bit as the control signal to the switching cells, the routing process can be done automatically.

In additional to the control signals, an *activity bit* is used to distinguish between idle input and intended 0-bound traffic. Usually, a global leading bit '1' will be used in contrast to the idle '0'. In each node in the network, the leading two bits of packet header are consumed as control signal, so as to launch a proper connection state. The node will then regenerate the leading activity bit and forward remaining part of the packet accordingly. Suppose the output address is in form of $d_1d_2...d_n$, and the guide of the bit-permuting network is given by $\gamma(1)\gamma(2)...\gamma(n)$, if we reformat the packet as shown in Figure 3.20, by adopting the distributed switching control mentioned above, the packet can be forwarded

properly **[21]**.

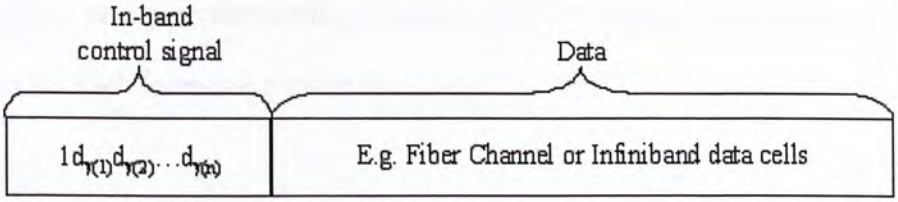


Figure 3.20 In-band control signal for self-routing in bit-permuting network

In particular, for monotonic increasing guide, the in-band control signal will be simply the activity bit together with the output address. For example, the guide of the network [: (34) : (14) : (24) :] is 3,1,2,4. By prepending the activity bit and the destination address bit to the packet in the specified order, i.e. $d_3d_1d_2d_4$, the network can automatically forward the packet to the right outbound interface.

Recall that a node in the switching network consumes two bits from the control signals to launch a corresponding connection state. It regenerates the activity bit and then shoots the remaining part of packet through the node. Since each node consumes the two leading control bits for forwarding decision, without alternating the remaining parts, no additional buffering is required. In this setting, data is not required to be written to and then read from the buffer memory, no memory operation is required, and therefore eliminating the performance limitation in buffering.

3.5.3 Multistage Interconnection of Self-routing Concentrators

As in nowadays technologies, switches are not 100% contention-free. To

improve system performance, alternative routing can be provided. However, it makes the routing decision at each node complex. Figure 3.21 suggests a simple alternative routing method called *statistical lines grouping*, which adopts a bundle of links for each inter-stage interconnection.

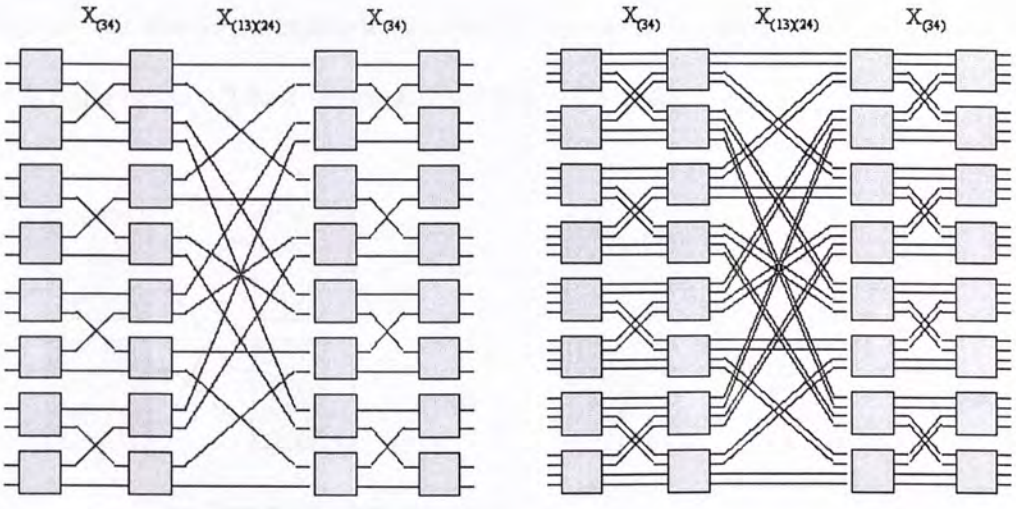


Figure 3.21 A 16x16 divide and conquer network and its 2-lined version

Intuitively, due to the presence of alternative paths, the overall blocking rate is diluted in proportion with the bundle size b. The larger the bundle size, the lower the overall blocking rate. When the blocking rate is so small that we can hardly experience once in system life time, we can consider the system as nonblocking in the practical engineering sense. Moreover, due to the close relationships with its unique-routing counterpart, the routing decision at each node becomes simpler and easier.

Note that when statistical line grouping is used, the underlying network is no longer a banyan-typed network. As the nodes we use now become 2bx2b, what we need for each node is a switch that can separate data into two sets of b

elements intending for the two different outgoing directions. A concentrator is a good building block for this task **[22][23]**.

An *m-to-n concentrator* is a device associated with some ordering, which divides the set of m elements into a large set of size n and a small set of size m-n. Figure 3.22 shows an example of concentrator with the dimensions 4-to-2, where each node being a 2-to-1 concentrator (or a 2x2 sorter).

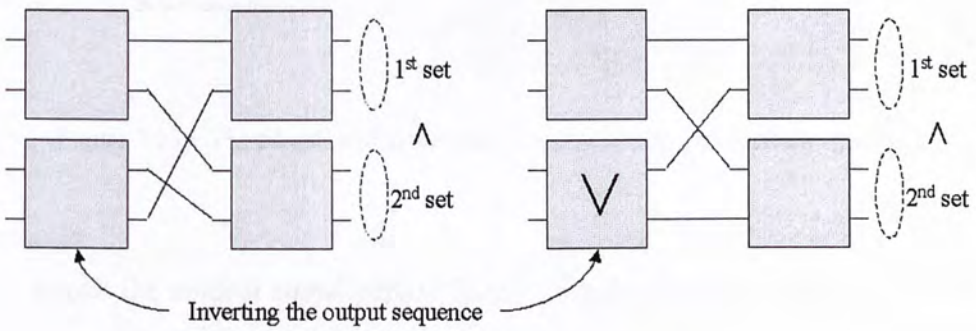


Figure 3.22 A 4-to-2 concentrator

Notice the fact that we always want the intended 0-bound signals to go to the top, while the intended 1-bound signals to go to the bottom. By sandwiching idle inputs, we can push the busy inputs to the right sides. Hence, if we impose the ordering for the concentrators in statistical line grouping as:

intended 0-bound < idle < intended 1-bound

we can replace the 2x2 nodes in a switching network with 2b-to-b concentrators in its b-line counterpart as illustrated in Figure 3.23. As concentrators can decide where a signal should go by examining the control signals, the concentration is also in some sense a self-service routing process.

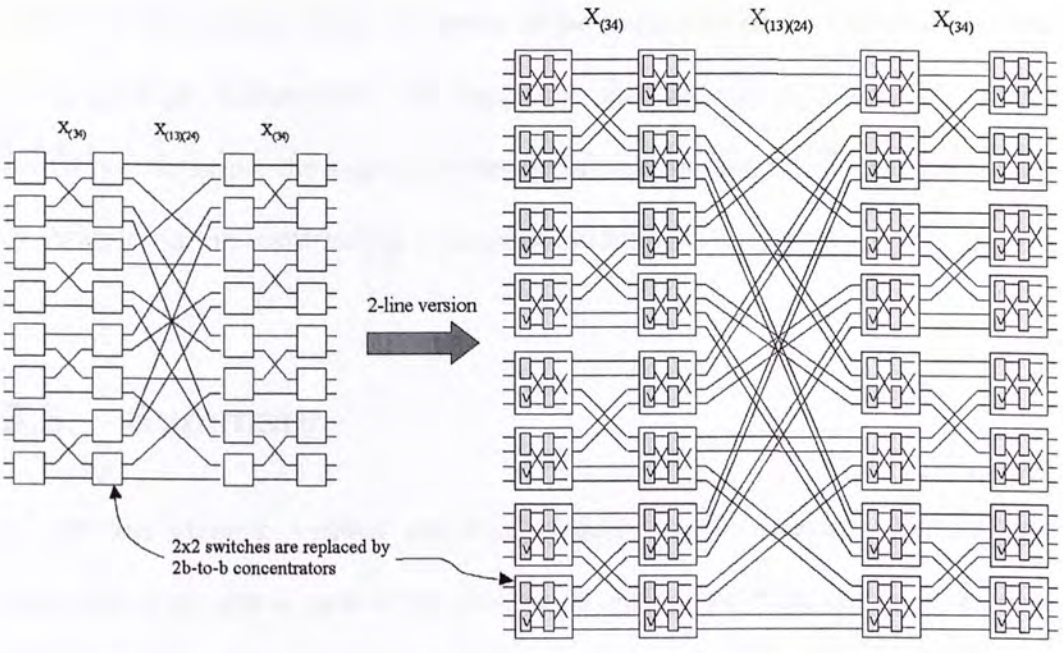


Figure 3.23 The adoption of self-routing concentrator in statistical line grouping

Recall the control signal pattern in self-routing algebraic switches defined in the previous section, where each node consumes two bits in deciding a connection state. The same approach can also be applied in statistical line grouping. The formation of control signal of a packet is depending on the guide of the underlying bit-permuting network, which is the same for b-line version networks. The prepended control signal is targeted at guiding the packet through the multistage bit-permuting networks. If the concentrator at each node consumes the leading two bits after concentration and then regenerates the leading activity bit, packet can be self-routed to its targeted destination without centralized control. The interstage control and per-stage concentration collectively demonstrate a two-level self-routing property in statistical line grouping.

The impact of self-routing property in algebraic switches is crucial. The elimination of centralized switching control helps to push the switching

bandwidth beyond the limit. Moreover, as the implementation of algebraic switch is technology independent, by replacing the underlying transmission and switching elements, the aggregate throughput can be upgraded. This gives us a qualitative start in constructing high-speed switches for the future.

3.6. Summary

In this chapter, various switching designs are investigated. Performance limitation is present in most of the designs. In order to meet the changing needs of the industry, scalable design is highly appreciated.

We have studied different proposals in switching designs, including ad-hoc designs, time division switches, shared buffer memory switches, crossbar switches and algebraic switches. Among the five types of designs, the later three proposals demonstrate a scalable performance, which is preferable in the deployment in storage networks. Besides, original amendments are proposed in this chapter, which are targeted on the switching aspect of storage networks. Suggestions on the deployment of different types of switches on the issue have also been made.

As stated early in this chapter, we are focusing on the case of a single-switch network. Obviously, if we only use a central switch as the core of device interconnection, it will easily be the bottleneck of performance, but for small and medium enterprise, this may be the most viable solution. Properly interconnection of switches can scale up the network to a higher bandwidth requirement.

4. Investigating Switching Issue in Storage Networks

In previous chapters, we have studied the needs of the industry and the switching technology for general-purpose storage networks. Switching and transmission bandwidth have been the two main factors incorporates are considering. Moreover, add-on functionalities also pay an important role in gaining market shares.

In deploying commercial switches to storage network uses, special considerations have to be made, including switch types, compatibility, add-on functionalities, etc. In this chapter, we will go through these issues and examine the suitability of various components in storage networking.

4.1 Choosing a Suitable Switch

In constructing our storage networks, a switch is usually used for interconnecting storage devices, no matter in the single-switch case or switched-fabric case. In the preceding chapter, various types of switches have been studied. Here we shall study their applications in storage networks.

Output queuing switches, such as shared-buffer memory switches, are highlighted at their high switching efficiency even at heavy network load.

However, the overall throughput is limited by the buffer I/O speed. For large-scale network, additional off-chip memory is needed for buffering. Externalizing of devices increases inter-chipset signaling and deteriorates the service of the switch. Therefore, output queuing switches are only good for medium-to-large scale network. But for nowadays storage networking sizes and technologies, output queuing switches are not good enough for the deployment.

As the counterpart of output queuing switches, input queuing switches in general give lower throughput because of its massive scheduling tasks. Moreover, as the buffers are specified by some input queues, its ability in absorbing network bursts is lower. The probability is relatively higher for an input queuing switch to overflow its buffer. Therefore, effective congestion control mechanism is important.

In the environment of storage networks, the initiator of data transfer is usually storage device. By nature, storage device can buffer any pre-sent data upon network congestion without additional buffering. In this situation, it will not be a great problem if any input buffer queue gets full. Moreover, as typical storage network nowadays usually fits in a building or a corporation, the network size is limited. The buffer overflow threshold can be set close to the limit, leading to a better buffer utilization. Therefore, input queuing switches are also suitable in medium-to-large size storage networks.

As storage networks are developing rapidly, conventional electronic-based switches, such as crossbar and shared-buffer switches, no longer satisfy the need of the industry. As the size and the bandwidth of these networks grow, more

powerful switches will be required. Algebraic switches can be an alternative. The examples given in the previous chapter help to develop media-independent switches, which pushes storage network throughput up to a higher level suitable for future applications, such as video-on-demand (VoD) and online cyber mall.

4.2 Quality of Service (QoS)

In application level, users may have different requirements on the underlying network services. For example, for video streaming, a late packet would carry no meaning as the frame has gone. Requirements like this bring out the issue of quality of service (QoS), where a maximum end-to-end delay is guaranteed. In particular, an implied maximum delay at each switching process is required.

A realtime packet has to arrive at its destination within a certain time. If it has already experienced certain amount of delay before entering a switch, it becomes more urgent. We can actually use weighted graphs as the network model to include the factor of delay.

For output queuing switches, in order to ensure the priority of some urgent packets, a special queue can be added at each egress port for these important packets. The switch can have different policies in handling the queues, such as weighted round-robin (WRR) queuing.

For input queuing switches, the preservation of QoS is done during scheduling. Since bipartite matching only gives the maximum matching, starvation may occur, which may cause severe delay for some packets. In

weighted environment, if we can raise the request weight of a packet upon waiting in the switch for a certain period of time, we can help meeting the maximum allowable delay for every packet. The strategy of raising the request weight of a packet is called aging. Similar technique has been demonstrated in the CPU scheduling of system software.

4.3 Multicasting

Nowadays, typical networks usually support various traffic modes, including unicast, multicast and broadcast. Unicast is the communication from a sender to a receiver. Due to its one-to-one communication nature, it is the most popular type of traffic in networks. Multicast refers to the communication from a sender to a group of receivers, and is often known as one-to-many communication. This type of traffic is suitable for information sharing, such as network conferencing and television broadcasting. Broadcast is, from its name, a kind of one-to-all traffic. This type of communication is unusual to users, but it is often adopted as a part of protocols, such as Dynamic Host Configuration Protocol (DHCP), Minimal Spanning Tree Algorithm, etc.

Multicast messages can be classified into two types, network layer or application layer. Multicast in network layer means that the data are transmitted by the transport layer (e.g. TCP/IP) to a multicast group, and the data is duplicated at network devices, such as switches or routers; multicast in application layer means, on the other hand, sending duplicated copies by the application explicitly, so as to allow multiple recipients. In current technology, multicast is usually done in application layer, solely because the underlying network does not have the

intelligence to support it.

Network layer multicasting is the issue of sending a same copy of packet to multiple recipients, so as to reduce effective switch throughput **[16]**. In review of the switching technologies discussed in the previous section, we will be proposing different approaches in expanding their functionalities to support multicasting. In this section, we will be looking into the network layer multicasting, which help us design multicast-ready switching components.

4.3.1 Crossbar Switch

As discussed in the previous chapter, a crossbar is a structured array of crosspoints such that every input port can be connected to any output ports. To forward packets to an output port, the switching scheduler needs to close the corresponding crosspoint. In order to prevent output contention, input buffering is used, which makes crossbar switch a typical example of input queuing switch.

In multicast situation, due to its special construction layout, we can easily send a packet to multiple output ports by closing multiple crosspoints simultaneously. Figure 4.1 gives a possible way in handling multicast traffic in crossbar switch.

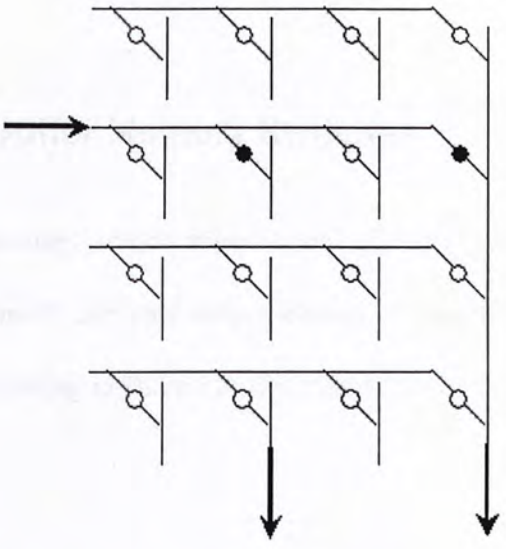


Figure 4.1 Multicast by the us of crossbar

To make multicasting works in crossbar switch, a slight modification to the scheduler will do the job. However, this kind of setting has a number of consequences. As mentioned before, output contention has to be dealt with by input queuing. In the multicast case, since the number of concerned egress ports increases, it is even harder to allocate an appropriate timeslot to send the data. The effect of HOL blocking may get more severe Moreover, as the message is distributed through the splitting of signal, as the fan-outs become large, the outgoing signal will become weak. Additional amplification is required.

Another alternative for multicasting is to require the ingress controller to buffer the packet, and explicitly send it to different output using unicast service. Despite the simplicity, it is not efficient in the sense that the switch needs to allocate more timeslot and buffer space for this service. The throughput is actually of similar magnitude of multicasting at application layer, and therefore this approach is not preferable.

4.3.2 Shared-Buffer Memory Switches

Shared buffer memory switch adopt a pool of buffer shared among different output ports, in contrast to separate output queues in generic switches. Due to the sharing nature, multicasting in shared buffer memory switches becomes not trivial **[16]**.

In order to simulate the multicast effect by similar techniques in unicast traffic, the ingress controller can be modified to enable the appending of duplicated copies of multicast packet at each of the output lists. Whenever a packet is decided to be dispatched, the packet can be treated the same as unicast message. However, as discussed in the previous section, it is not a good approach in the sense of switching as it increases the loading of the switch. Extra memory space is needed in dealing with multicast traffic. Due to the similarity to application layer multicasting, this approach is not popular in the industry.

To improve the way of serving multiple recipients, we target at keeping only a single copy of the incoming packet. Recall the structure of a shared buffer memory switch, where each output port is associated with an egress controller. The job of an egress controller is to keep a linked list of intended outgoing packets, discharge them one by one physically to the port, and then delete the packet to free the memory for future use. If only a single copy of the packet is kept for multiple ports, the memory address of the packet can still be put to the linked lists of each intended output port. However, we need to make sure the packet is not deleted before all the egress controllers have finished using it.

One of the possible ways is to keep a counter for every multicast packet. Instead of buffering the plain packet in the shared buffer, a special structure consisting of a multicast counter and the packet content is used as shown in Figure 4.2. Whenever a multicast packet comes in, the queuing controller store the packet content in the structure in the memory, append the address of the structure to each outgoing linked list, and then initiate the multicast counter associate with this packet as the total number of intended outputs. For every discharging of concerning packets, the corresponding counter is decrement by one. A packet structure will only be deleted from the memory when the counter reaches zero.

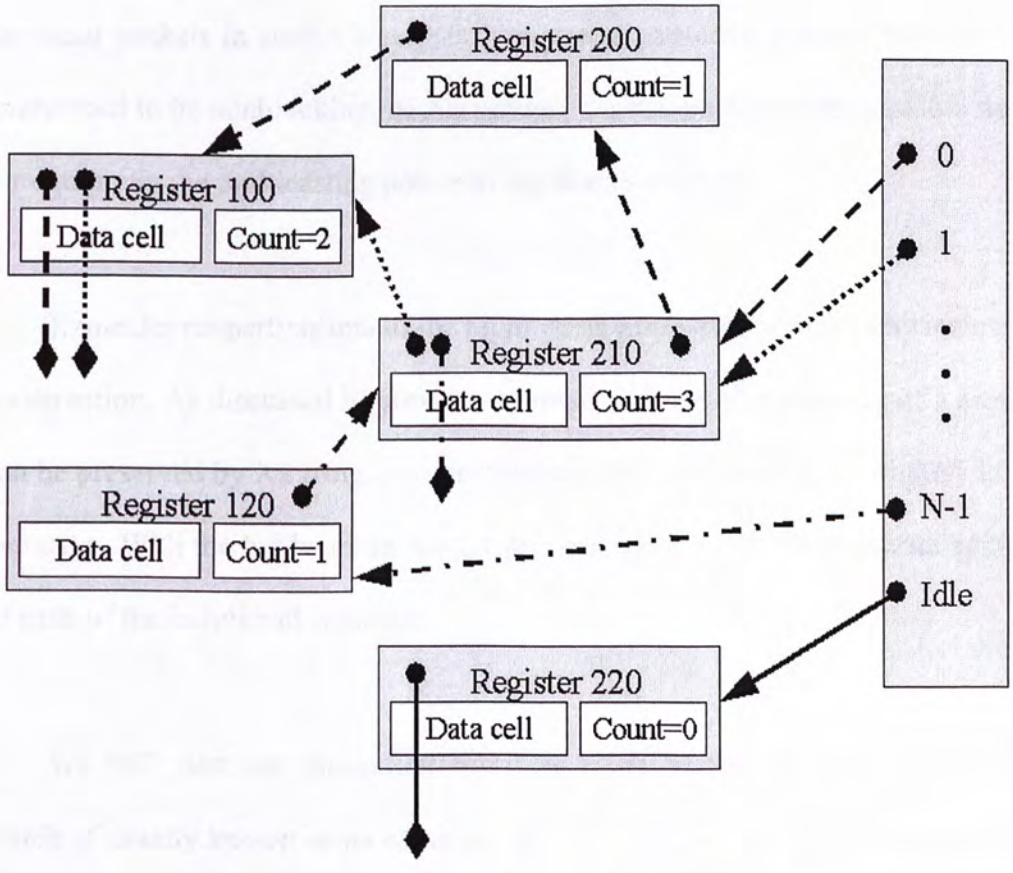


Figure 4.2 Multicasting in shared buffer memory switch

By this strategy, only a single copy of the incoming packet is required in the memory, implying the good utilization of buffer memory. We can guarantee the required packet will not be freed from memory too early, and hence the accuracy of the switch. This approach can also be generalized to unicast traffic by initializing the corresponding counter as one, making the switch capable to meet major traffic requirement of network users.

4.3.3 Algebraic Switch

Conditionally nonblocking properties are one of the major characteristics of algebraic switches. With the absence of output contentions, if we can pre-arrange the input packets in such a way that they obey a particular pattern, the switch is guaranteed to be nonblocking. In this section, we will be discussing possible ways for equipping the multicasting power to algebraic switches.

Expander property is one of the highlighted properties preserved by recursive construction. As discussed in previous chapter, the expander property of a switch can be preserved by X2 recursive construction, and hence building infinitely large expander. With the hardware in hand, the crucial part is now the software control of each of the individual expander.

We will start our discussion from the basic component, a 2x2 expander, which is usually known as an expander cell. It includes four different connection states, as shown in Figure 4.3, which work together to provide multicast switching services.

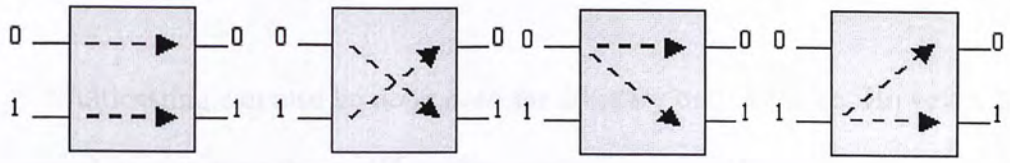


Figure 4.3 The four different connection states of an expander cell

In launching any of the connection states, we need to take care with the intended destination of both ingress ports. For each ingress port, there are four possible cases of forwarding request, namely *idle*, *intended 0-bound*, *intended 1-bound* and *bi-cast*. Therefore, we can construct a table for all the 4^2, i.e. 16 possible request combinations, as shown in Table 4.1.

Input 0 \ Input 1	Idle	0-bound	1-bound	Bi-cast
Idle	Any	Cross	Bar	1-bicast
0-bound	Bar	*Any for Contention*	Bar	*Any for Contention*
1-bound	Cross	Cross	*Any for Contention*	*Any for Contention*
Bi-cast	0-bicast	*Any for Contention*	*Any for Contention*	*Any for Contention*

Table 4.1 The connection table of an expander cell

Even when there are some uncertainties in launching connection states for the cases of contention, the situation can be avoided by traffic prearrangement according to the Expander Theorem. Therefore, under this special traffic pattern, connection state can be decided by the intended forwarding preferences, which can be prepended to the packet as a form of self-routing control data, as discussed in the previous chapter.

Multicasting can also be done even for arbitrary output traffic. However, the overhead in preparing these self-routing control signals is huge. In order to build practical multicast switches, we can make use of the self-routing properties to a rectangular set of output addresses **[21]**. Note that the set of all n-bit output addresses can be generalized in a form of $\{0,1\}x\{0,1\}x...x\{0,1\}$. A rectangular set of addresses is the subset $S_1xS_2x...xS_n$, where S_j is a non-empty subset of $\{0,1\}$. As an example, the rectangular addresses $\{0\}x\{0,1\}x\{0\}x\{1\}$ represent the group of addresses $\{0001, 0101\}$.

For the self-routing of a rectangular set of output addresses, the control signal of the packet is simple. Let Q_j be the set of the four possible forwarding preferences of each individual ingress port, each of those of a size of two bits. If the packet is going to be forwarded through a banyan-type network with the guide $\gamma(1)\gamma(2)...\gamma(n)$, the control header will simply be $Q_{\gamma(1)}Q_{\gamma(2)}...Q_{\gamma(n)}$, as shown in Figure 4.4.

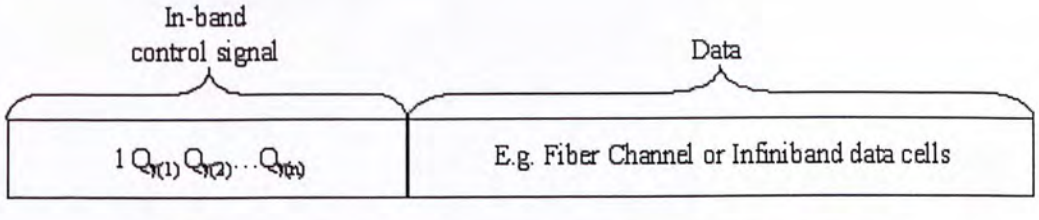


Figure 4.4 In-band control signals for multicast banyan-type network

The control signal is again containing an activity bit to distinguish between idle inputs. Whenever a packet comes in, a node in the network consume three bits from the control header. By the information given from the two-bit

connection preference, a proper connection state should be launched, and the activity bit is then regenerated and sent out together with the packet.

Since the activity bit is discarded upon receiving of packet data, the node needs only read two bits out of the first three bits for launching a connection state. As the leading control bits are to be consumed by the node and are not to be regenerated, the node needs only a single bit buffering in the multicasting process (as shown in Figure 4.5), which reduces the total buffer size dramatically. For monotonic increasing guide, the in-band control signal will be simply be the activity bit together with the output addresses of both output ports.

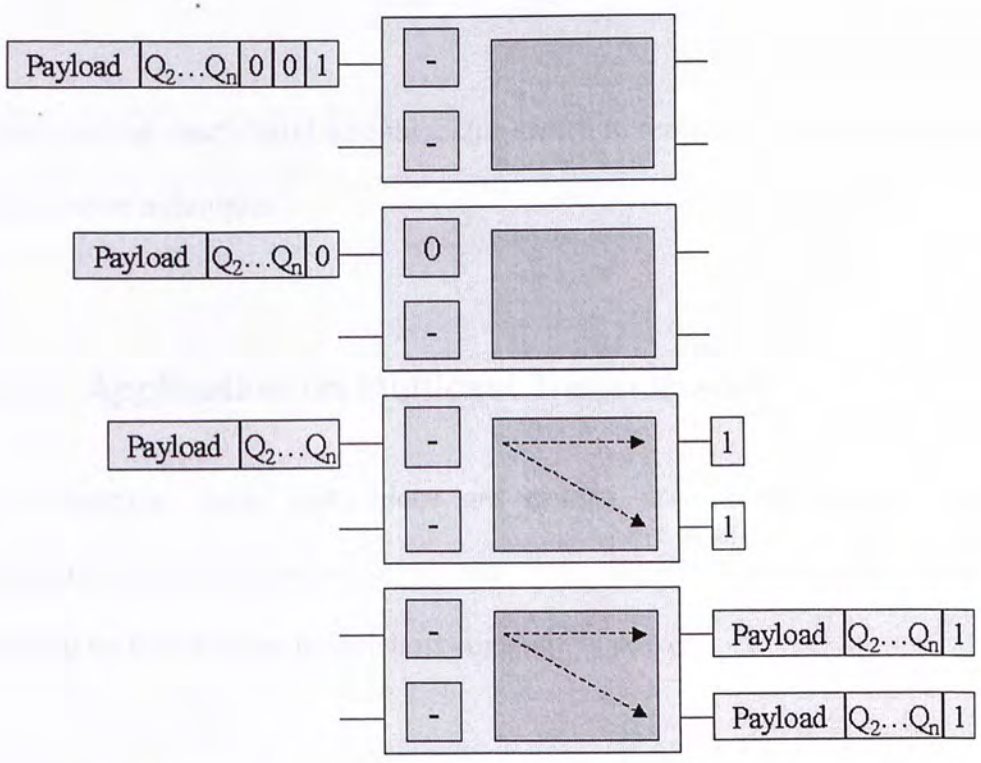


Figure 4.5 Single bit buffering for multicasting to a rectangular set of addresses

With the assistance of statistical line grouping, we can build an engineering nonblocking switch in combination with traffic pre-arrangement. Consider a *m*x*m*

switch with a bundle size of b, if every of the ingress port transmit only b/m cells in a timeslot, we can guarantee that even in extreme case, where all of the input cells are targeted at a specific output port, there will only be $(b/m)*m$ cells reaching it, which does not exceed the bundle size limit b. Under this limitation, we can guarantee that the switch is nonblocking.

As an example, for the 16x16 divide and conquer switch shown in Figure 3.21, if a bundle size of 64 is used, with a fixed cell length of 64 bytes. If we set our timeslot into 2μs, this switch can reach a bandwidth of:

$$Bandwidth = \frac{64bytes * 8bits}{2\mu s} \cdot \frac{64}{16} = 1k\ bits / \mu s = 1Gb / s$$

Hence, we can easily build a nonblocking switch to reach the bandwidth of 1Gb/s by the above techniques.

4.3.4 Application on Multicast Transmission

Nowadays, more applications are making use of multicasting traffic. Application in storage networks can also make use of the multicasting function provided by the switches to enrich its competitive power.

In multicasting, the same piece of information is transmitted to various clients. Applications like video broadcasting and NetTV would be good examples for the use of multicasting. The file data can be stored in storage network, and is sent out through an outgoing gateway or interface.

Apart from the one-way video distribution, interactive application can also be a good example in multicasting. An online cyber mall is a good example. The application will be broadcasting general background information to clients, such as the virtual reality display information. Depending on specific client's instruction, addition unicast message may be sent, such as online purchasing instruction. Information of the cyber mall is usually obtained from separated shops or companies, where storage network can be a suitable back-end storage strategy.

Another highlighted application for storage network is distance backup. In current technology, most of the backup process is in form of batch processing, where real-time data may lose in disaster. Distance backup is a real-time data storing strategy with multiple copies distributed in various physical locations. In storage networks, clients can store any information to any remote networked storage device. By adopting the benefits of multicasting, data storing can be split into multiple network data streams targeted at different physical backup centers. By this method, crucial information can be stored and scattered in different locations, which can handle different disasters. However, since this application deals with files or data, integrity is an important issue, and hence reliable multicasting is required.

4.4 Load Balancing Mechanism

In the world of storage networks, multiple servers are usually available in redundant to provide fault tolerance, but most of the time, all of the servers are

functioning properly. However, the traffic load for different servers may vary. To achieve a better overall performance, a load balancing mechanism among servers is important.

A simple method for load balancing is to compress the traffic from, say thousands of clients to a hundred servers. As the active time for each client is limited, the server-to-client ratio can be toned to make it work at an acceptable performance. In this setting, clients can get served by the servers in either a round-robin manner or by a FIFO.

However, as you may notice, the design does not give any guarantee on services. A possible modification is to help the packet in selecting a suitable server from a pool of servers. The introduction of *load resolver* gives a possible solution to the problem.

A load resolver is a tailor-made device to give out server address to each incoming packet. Assuming that all of data requests in the storage network are small in size, typically within a packet size, the command will not be segmented into smaller cells. Hence in making forwarding decision, we can treat the packets independently, and can possibly be routed to different servers. By installing the resolver in the network as shown in Figure 4.6, it can help diverting packets to different servers, and in a long run, balancing traffic loading.

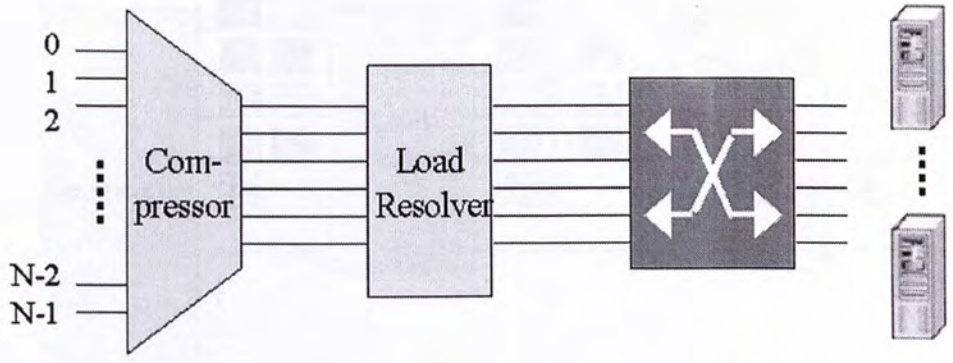


Figure 4.6 Adoption of load resolver in load balancing

When the load resolver is used in conjunction with a compressor, all active connections will be concentrated at a certain ingress port of the resolver. Note that the traffic has been concentrated by the compressor, servers that are far from the concentration point will become less busy. The resolver aims at shifting the traffic pattern by re-centering the concentration point of traffic in a round robin manner.

Figure 4.7 shows a practical use of load resolver in offloading the servers. Instead of letting the traffic going directly to the servers, the load resolver acts as a traffic distributor, giving out server addresses in round-robin manner so that packets can go to any less-loaded servers. Moreover, due to the bursty nature of network traffic, the resolver can offload the heavy traffic form a particular ingress port to different servers, achieving a higher throughput in a long run.

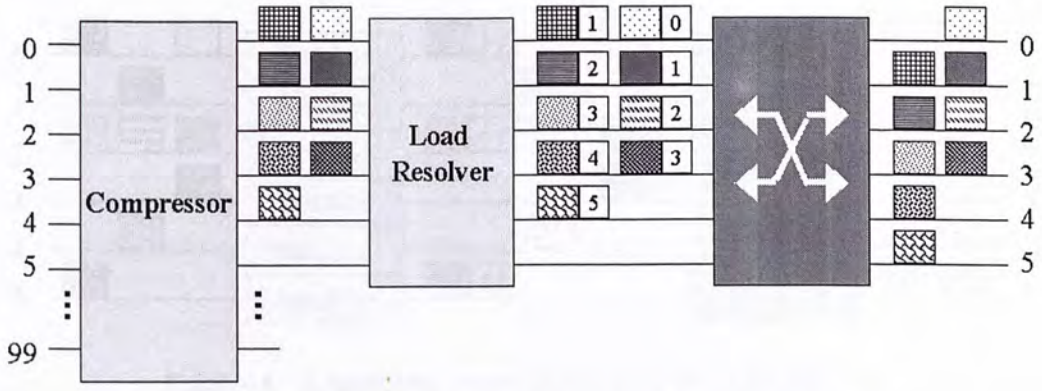


Figure 4.7 Example of using load resolver for load balancing

As the load resolver distributes addresses one at a time, we can be sure that no output contention is available. The switching fabric here can be very simple, or adopt the algebraic switch to push up the switching performance. Due to the regular pattern of the whole system, we can also simulate the effect by configuring a crossbar switch with only diagonal crosspoints connected, and the connection state shifts in each timeslot. Because of the absence of output contention, the performance of crossbar switch is very high.

However, as the number of servers increase, the load resolver becomes a performance bottleneck, as it requires centralized control. In view of the problem, a distributed version of resolver can be used.

Figure 4.8 shows a distributed load resolver. Each input node is connected to an independent, distributed end-device, which gives out server address in round-robin manner (instead of a centralized one). The incoming packets will be routed to different servers at different time slots.

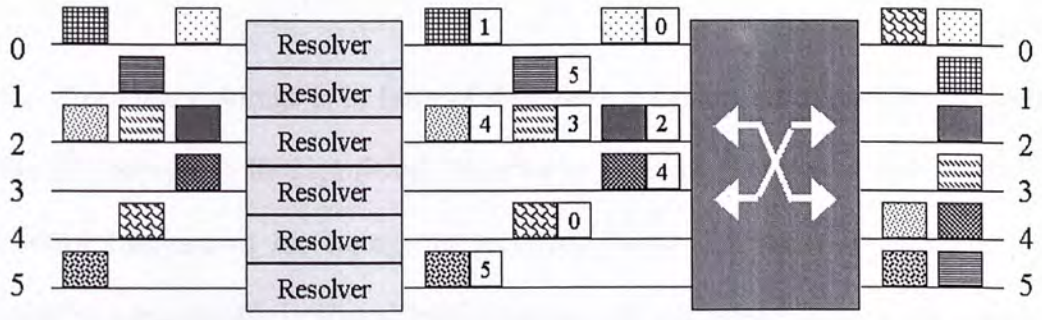


Figure 4.8 A distributed version of load resolver in load balancing

Due to the bursty nature of network traffic, load fluctuation will usually occur in one or some of the inputs, but not all. A distributed load resolver for each input can undoubtedly divert them into different servers, and therefore achieve a more balanced loading. The loading in this setting may vary in a relative short period, but in time average, it can perform quite well.

However, despite the simplicity of the setting and the distributive nature of the devices, output contention is an issue to this setting. Since all the devices are operating independently, the same address may be arranged by more than one device. The switching fabric must be able to handle the potential routing problems, such as buffering and queuing issues discussed in previous chapter.

4.5 Optimization on Storage Utilization

When we focus into the problem of storage, we are mainly focusing our studies in memory size and memory bandwidth. Adding more storage devices can only solve the problem in storage size, but not the memory bandwidth. To improve both of them, we need a parallel-processed array of storage. Appropriated data storage strategy is important.

Secondary storage is in form of slow devices. When we are assessing them, the I/O delay can be significant. In order to leverage these overheads, suitable control mechanism on storage is required. Recall the ways in speeding up switching bandwidth in shared buffer memory switch, where parallel memory or distributed storage can be useful, given that each block size is small enough. Similar approach can be deployed in secondary storage for performance improvement.

For large data files, we may be required to store it in distributed devices so as to reduce its I/O latency. The storage devices consist of multiple blocks, and each block is of a fixed size. If a total number of n storage devices are used with a block size of b. If data files are sufficient times larger than the product of nb, data storage strategies can efficiently increase the I/O throughput of the system, and hence the performance of the storage networks.

A possible usage of this technique is video distribution or cyber mall, where visual information has to be read and distributed to various users simultaneously. Consider a storage bank of video files, which are of the dimension of megabytes. The assumption for small enough block size is valid. Parallel array of disks can be used for simultaneous data achievement, so as to lower the I/O latency. Moreover, distributed data stores can also be employed, where data blocks are achieved from various disks in round-robin manner, so that system bottleneck in I/O buses and memory can be diversified.

This technique is of significant importance. In the world of storage networks,

the slowest operating devices are secondary storage. Without efficient way in speeding up the assessing time, even if we can push the network bandwidth to the edge of technology, the overall system performance will not be improved.

4.6 Summary

Various switches can be used in building large storage networks, but which type is the best fit into the design is an important issue. This chapter raises various switching concerns in selecting a right component. Improvement has been proposed to enhance the functionalities of storage networks. Different aspects of storage networks have been studied, including switching, traffic pattern, and storage pattern. With the proposals presented, the performance as well as the functionalities in storage networks can be enriched.

5. Conclusion and Summary of Original Contributions

In this thesis, we have discussed various proposals of switching in the viewpoint of storage networks. As the wide development of switched-fabric storage networking, the role of switching is definitely important. Without an evolution on switching technologies, it will soon become the bottleneck of most of the network applications. In view of the urgent needs of the industry, we have proposed various proposals in pushing up switching bandwidth.

Experts from various official bodies have been looking into the future of storage networking and forecasted the migration of storage networks with IP standards **[10][14]**, which has been developed for years. Being a widely deployed mature technology, IP networking in storage can provide a feasible solution to different groups of users. On the other hand, tailor-made protocols, such as InfiniBand is also gaining importance in the market. The co-operations among various technologies can definitely benefit the growth of storage networking.

Originated work has been presented in this thesis on the linkage between storage networking and switching. Besides, there are sections in each chapter that present original ideas from myself. These sections include the parallel memory array and distributive storage in shared-buffer memory switches (3.3.1, 3.3.2), scheduling of crossbar switches (3.4.2) and the switching investigations and

designs for specific applications in storage network (Chapter 4). The switching technologies proposed in this thesis can not only be used in storage networking, but also be used into general-purpose network.

This thesis is intended for a bridge between storage networks and switching technologies.

Reference

1. Storage Network Industry Association. *iSCSI: New Business Benefits and Solutions*. [Online] Available: http://www.snia.org/tech_activities/ip_storage
2. Storage Network Industry Association. *Backup and Recovery of Storage Networks*. [Online] Available: http://www.snia.org/education/tutorials/Backup-Restore_of_Storage_Networks.pdf.
3. Gupta, M. *Storage Area Network Fundamentals*. Cisco Press, 2002, USA.
4. Storage Network Industry Association. *Clearing the Confusion: A Primer on Internet Protocol Storage*. [Online] Available: http://www.snia.org/education/tutorials/ip_storage.pdf.
5. Storage Network Industry Association. *Directory of Storage Networking*. [Online] Available: http://www.snia.org/education/dictionary
6. Storage Network Industry Association. *Storage for Networking Professionals*. [Online] Available: http://www.snia.org/education/tutorials/Storage_for_Networking_Professionals.pdf.
7. Storage Network Industry Association. *Assessing and Designing Storage Network Infrastructures*. [Online] Available: http://www.snia.org/education/tutorials/Storage_Infrastructure.pdf.
8. Fibre Channel Industry Association. *Executive Summary*. [Online] Available: http://www.fibrechannel.org/OVERVIEW/index.html
9. Adaptec Inc. *Fibre Channel Frequently Asked Questions*. [Online] Available: http://www.adaptec.com/worldwide/product/markeditorial.html?prodkey=fibrechannel_faqs.

10. Digital Networks. *Techniques for Storage Networking Over IP*. [Online] Available: http://www.dnpg.com/newsevents/whitepapers/IP_SANs.doc.
11. Dell Computer Corporation. *InfiniBand Architecture: Next-Generation Server I/O*. [Online] Available: http://www.dell.com/us/en/gen/topics/vectors_2000-infiniband.htm, 2000
12. InfiniBand Trade Association. *InfiniBand Architecture Overview*. [Online] Available: http://www.infinibandta.org/events/past/it_roadshow/overview.pdf
13. iSCSI Storage. *IP storage: A review of iSCSI, FCIP, iFCP*. [Online] Available: http://www.iscsistorage.com/ipstorage.htm, 2003.
14. Storage Networking Industry Association. *The SNIA vision: Setting the pace for the Industry*. [Online] Available: http://www.snia.org/data/resources/presentations/20010607_FCIA_Maastricht_Kranz.ppt, 2001.
15. McKeown N., *Lecture Notes: Packet Switch Architectures*. Stanford University, 2003, USA.
16. Aweya J., *IP Router Architectures: An Overview*. Nortel Networks, Canada.
17. Iyer S., Kompella R. and McKeown N., *Designing Packet Buffer for Router Line cards*. Computer Systems Laboratory, Stanford University, USA.
18. Chrysos N. and Katevenis M., *Transient Behavior of a Buffered Crossbar Converging to Weighted Max-Min Fairness*. [Online] Available: http://archvlsi.ics.forth.gr/bufxbar/bufxbar_chrys_02-08.pdf
19. Patel R., *Lecture Notes: Internetworking and Distributed Systems Laboratory*. University of Southern California, 2003, USA.
20. Stengel B., *Theory of Algorithms: Bipartite Matching*. Department of Mathematics, London School of Economics & Political Science, 2003, U.K.
21. Li S.-Y. R., *Algebraic Switching Theory and Broadband Application*. Academic Press, 2000, Hong Kong.

22. Li S.-Y. R and Lam W., *ATM Switching by Divide-and-Conquer Interconnection of Partial Sorters*, Microprocessors and Micro Systems, Vol. 22, pp. 579-587, May 1999.

23. Li S.-Y. R., *Switching by Multi-stage Interconnection of Concentrators*, USA Patent No. 6591285, June 24, 2003.